IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

APPEAL BRIEF

RECEIVED
FEB 2 7 2003
Technology Center 2100

Inventors	:	Oehrke, *et al.*
Serial No.	:	09/021,466
Filing Date	:	February 10, 1998
Title	:	SURVIVABLE AND SCALABLE DATA SYSTEM AND METHOD FOR COMPUTER NETWORKS
Group/Art Unit	:	2153
Examiner	:	Kupstas, Tod
Docket No.	:	1177

Commissioner for Patents
Washington, D.C. 20231

Sir:

In accordance with the provisions of 37 C.F.R. §1.192, Applicant submits this Appeal Brief in support of the Notice of Appeal filed on December 11, 2002 and received in the U.S. Patent and Trademark Office on December 20, 2002.

I. REAL PARTY IN INTEREST

The real party in interest in the present appeal is the assignee, Sprint Communications Company, L.P. The assignment was recorded at Reel 8976, Frame 0965 of the U.S. Patent and Trademark Office records.

Certificate of Mailing Under 37 C.F.R. 1.8
I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to Assistant Commissioner for Patents, Washington, D.C. 20231, on:
Date: February 20 2003
Signature: Merry Longsworth
Printed Name: MERRY LONGSWORTH

The Director is hereby authorized to charge any additional amount required, or credit any overpayment, to Deposit Account No. 19-4409.

II. RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences.

III. STATUS OF CLAIMS

Claims 79-115 are pending in the application. Claims 1-78 have been canceled. Claims 79-115 stand finally rejected, as follows: claims 79-80, 82-88, 91-95, 98, 104, 106 and 108-114 are rejected under 35 U.S.C. §102(e) as being anticipated by U.S. Patent No. 5,699,503 to Bolosky et al.; claims 81, 89, 96, 100, 105, 107 and 115 are rejected under 35 U.S.C. §103(a) as being obvious over U.S. Patent No. 5,699,503 to Bolosky et al.; and claims 90, 97, 99 and 101-103 are rejected under 35 U.S.C. §103(a) as being obvious over U.S. Patent No. 5,699,503 to Bolosky et al. in view of U.S. Patent No. 4,914,570 to Peacock. The present appeal is directed to claims 79-115.

IV. STATUS OF AMENDMENTS

Claims 79-115 have not been amended. These claims are reproduced in Appendix A attached hereto.

V. SUMMARY OF THE INVENTION

The present invention is directed to a system and method for providing network processing and stored data access that is configured to be fully scalable and/or fully survivable. Two different embodiments of the invention are shown in Figs. 1 and 2 of the application, attached hereto as Appendix B. As can be seen, at least one server (also referred to as an

application processor) is provided that is operable to process user requests. The server is connected to a switch, which is in turn connected to at least one data storage device.

In one aspect of the invention, the system is fully "scalable" in the sense that additional servers can be added to the system as demand for a particular application increases, <u>without adding additional data storage devices</u>. Conversely, servers can be removed from the system without removing data storage devices. In a similar manner, additional data storage devices can be added to the system as storage requirements for a particular application increase, <u>without adding additional servers</u>. Conversely, data storage devices can be removed from the system without removing servers. Thus, the system is scalable to increase or decrease server capacity without changing the data storage capacity, and/or is scalable to increase or decrease data storage capacity without changing the server capacity.

In another aspect of the invention, the system includes at least two servers that apply substantially the same application(s) when processing user requests, and at least two data storage devices that contain substantially identical data. The system is fully "survivable" in the sense that, if any one of the servers fails, user requests can be processed by any of the other servers in the system that are operable. Likewise, if any one of the data storage devices fails, substantially identical data can be retrieved from any of the other data storage devices that are operable. Thus, the system is survivable and able to process user requests in the event of a failure of any one of the servers, and is survivable and able to retrieve data in the event of a failure of any one of the data storage devices.

Claims 79-85, 89-97, 102, 104-108, 110 and 112 are directed to the scalability aspect of the claimed invention. Claims 98-101 and 114-115 are directed to the survivability aspect of the

claimed invention. Claims 86-88, 103, 109, 111 and 113 are directed to both the scalability and survivability aspects of the claimed invention.

VI. ISSUES

The issues on appeal are as follows:

A. Whether claims 79-80, 82-88, 91-95, 98, 104, 106 and 108-114 are unpatentable under 35 U.S.C. §102(e) as being anticipated by U.S. Patent No. 5,699,503 to Bolosky et al.

B. Whether claims 81, 89, 96, 100, 105, 107 and 115 are unpatentable under 35 U.S.C. §103(a) as being obvious over U.S. Patent No. 5,699,503 to Bolosky et al.

C. Whether claims 90, 97, 99 and 101-103 are unpatentable under 35 U.S.C. §103(a) as being obvious over U.S. Patent No. 5,699,503 to Bolosky et al. in view of U.S. Patent No. 4,914,570 to Peacock.

VII. GROUPING OF THE CLAIMS

With respect to the rejection stated in Issue A, claims 79-80, 82-85, 91-95, 104, 106, 108, 110 and 112 stand or fall together; claims 98 and 114 stand or fall together; and claims 86-88, 109, 111 and 113 stand or fall together. As discussed in Section VIII.A.4 below, these three different groups of claims are separately patentable.

With respect to the rejection stated in Issue B, claims 81, 89, 96, 105 and 107 stand or fall together; and claims 100 and 115 stand or fall together. As discussed in Section VIII.B.3 below, these two different groups of claims are separately patentable.

With respect to the rejection stated in Issue C, claims 90, 97, and 102 stand or fall together; claims 99 and 101 stand or fall together; and claim 103 stands or falls alone. As

discussed in Section VIII.C.3 below, these three different groups of claims are separately patentable.

VIII. ARGUMENT

A. Applicant's Claims are not Anticipated by Bolosky

The Examiner rejected claims 79-80, 82-88, 91-95, 98, 104, 106 and 108-114 under 35 U.S.C. § 102(e) as being anticipated by U.S. Patent No. 5,699,503 to Bolosky et al. ("Bolosky"), attached hereto as Appendix C. However, as discussed below, Bolosky does not disclose the survivability and/or scalability aspects of the claimed invention.

1. The Bolosky Disclosure

Bolosky discloses a media server system, such as a video-on-demand system, in which video image sequences (*e.g.*, a movie) are transmitted from the system to subscribers in response to user requests. The system includes a controller (*e.g.*, controller 16 of Fig. 2) and a plurality of subsystems (*e.g.*, subsystems 18A, 18B and 18C of Fig. 2). Bolosky, col. 5, l. 61 to col. 6, l. 10. Each subsystem comprises a single microprocessor (*e.g.*, microprocessor 20A of Fig. 2) and one or more data storage devices (*e.g.*, data storage devices 22A and 24A of Fig. 2). Bolosky, col. 6, ll. 11-19. In operation, the controller cooperates with the microprocessor of each of the subsystems to schedule the transmission of video image sequences stored on the data storage devices to the subscribers. Bolosky, col. 6, ll. 20-23.

The video image sequences are stored on the data storage devices of all of the subsystems by dividing them into sequential blocks of data and "striping" them across the primary portions of the data storage devices. Bolosky, col. 6, ll. 40-43. "Striping" refers to the method in which a first block of data is stored on a first data storage device and each sequentially following block of

data is stored on the next sequential data storage device. Bolosky, col. 6, ll. 46-49. When reaching the last data storage device, the next block of data wraps around and is stored on the first data storage device of the first subsystem. Bolosky, col. 6, ll. 49-51. This continues until all the blocks of data are stored across all of the data storage devices. Bolosky, col. 6, ll. 51-53.

After the blocks of data are stored on the primary portions of the data storage devices, "declustered mirroring" is used to store a copy of the same data on the secondary portions of the data storage devices. Bolosky, col. 6, ll. 57-61. An example of "declustered mirroring" is shown in Fig. 3D. In this example, a block of data stored on the primary portion of a data storage device of a first subsystem (*e.g.*, "Subsystem 1, SD1, Block A") is divided into first and second sub-blocks of data, wherein the first sub-block of data is stored on the secondary portion of a data storage device of a second subsystem (*e.g.*, "Subsystem 2, SD3, Sub-Block A1") and the second sub-block of data is stored on the secondary portion of a data storage device of a third subsystem (*e.g.*, "Subsystem 3, SD5, Sub-Block A2"). Bolosky, col. 9, lines 14-34. In the event of a failure of the first subsystem, the first and second sub-blocks of data can be transmitted from the second and third subsystems to the subscribers. Id.

2. Bolosky Does Not Disclose the Scalability Aspect of the Claimed Invention

Independent claims 79, 82, 91, 104 and 106 (and dependent claims 80, 83-88, 92-95 and 108-113), which are directed to the scalability aspect of the claimed invention,[1] each require that a server operates independently of a data storage device so as to permit the addition (or removal) of a server without the addition (or removal) of a data storage device (*e.g.*, as demand for a particular application increases or decreases). Bolosky does not disclose or suggest this

[1] Dependent claims 86-88, 109, 111 and 113 are also directed to the survivability aspect of the claimed invention, discussed in Section VIII.A.3 below.

limitation. Rather, Bolosky discloses multiple subsystems that each include a microprocessor tied to one or more data storage devices. Within each subsystem, the microprocessor does not operate independently of the data storage devices. As such, it is not possible to: (1) add a new microprocessor without also adding one or more new data storage devices to create a new subsystem; or (2) remove a microprocessor of a subsystem without also removing the data storage devices of the subsystem.

The Examiner repeatedly cites to three different portions of the Bolosky specification to support his contention that Bolosky discloses a system that is fully scaleable. However, none of these portions disclose a server that operates independently of a data storage device to permit the addition (or removal) of a server without the addition (or removal) of a data storage, as claimed by Applicant:

1. Bolosky, col. 8, l. 38 to col. 9, l. 34
   This portion of the Bolosky specification discloses that a declustering number (*i.e.*, the number of sub-blocks of data stored across multiple data storage devices) can be chosen so as to tolerate the failure of more than one data storage device or subsystem. An alternative embodiment of "declustered mirroring" is also disclosed, wherein the burden of performing failure mode processing is spread across a larger number of data storage devices than in the preferred embodiment.

2. Bolosky, col. 5, l. 61 to col. 6, l. 23
   This portion of the Bolosky specification discloses that, although the preferred embodiment describes three subsystems, a larger number of subsystems will typically be employed. This portion also discloses that while each subsystem of the preferred embodiment includes a single microprocessor that is responsible for controlling two data storage devices, each subsystem may alternatively include one data storage device or more than two data storage devices.

3. Bolosky, col. 7, ll. 4-28
   This portion of the Bolosky specification discloses that the declustering number may vary, and that a higher declustering number can be chosen to: (1) lessen the burden of failure mode processing by any one data storage device, and (2) reduce the bandwidth reserved for failure mode processing.

All of the above portions of the Bolosky specification are directed to the system's capability of varying the number of data storage devices within a subsystem to enhance fault tolerance. Nowhere does Bolosky disclose a microprocessor that operates independently of a data storage device to permit the addition (or removal) of a microprocessor without the addition (or removal) of a data storage device. Rather, in Bolosky, each microprocessor is tied to particular data storage devices (albeit the number of data storage devices may vary).

Thus, independent claims 79, 82, 91, 104 and 106 (and dependent claims 80, 83-88, 92-95 and 108-113) are not anticipated by Bolosky and should be allowed.

3. Bolosky Does Not Disclose the Survivability Aspect of the Claimed Invention

Independent claims 98 and 114 (and dependent claims 86-88, 109, 111 and 113), which are directed to the survivability aspect of the claimed invention,[2] each require first and second (or a plurality of) data storage devices that <u>each</u> store <u>substantially the same data</u> such that, in the event of a failure of <u>any one</u> of the data storage devices, the data is accessible from <u>any other</u> of the data storage devices that are operable. Bolosky does not disclose or suggest this limitation. Rather, in Bolosky, each data storage device stores <u>different blocks of data</u> such that no <u>one</u> data storage device stores substantially the same data as <u>any other</u> data storage device.

The Examiner argues that the "declustered mirroring" process of the Bolosky system discloses this limitation. Not true. As can be seen in Fig. 3D, each data storage device stores a completely different set of data -- SD1 stores Block A and Sub-Blocks I1 and G2, SD2 stores Block B and Sub-Blocks F1 and D2, etc. As such, the Bolosky process of storing different blocks of data on each data storage device is directly contrary to the claimed invention.

---

[2] Dependent claims 86-88, 109, 111 and 113 are also directed to the scalability aspect of the claimed invention, discussed in Section VIII.A.2 above.

Thus, independent claims 98 and 114 (and dependent claims 86-88, 99-101, 103, 109, 111, 113 and 115) are not anticipated by Bolosky and should be allowed.

4. Different Groups of Claims are Separately Patentable

Claims 79-80, 82-85, 91-95, 104, 106, 108, 110 and 112 are directed to the scalability aspect of the claimed invention, and are not anticipated by Bolosky for the reasons discussed in Section VIII.A.2 above. Claims 98 and 114 are directed to the survivability aspect of the claimed invention, and are not anticipated by Bolosky for the reasons discussed in Section VIII.A.3 above. Claims 86-88, 109, 111 and 113 are directed to both the scalability and survivability aspects of the claimed invention, and are not anticipated by Bolosky for the reasons discussed in Sections VIII.A.2 and VIII.A.3 above.

B. Applicant's Claims are not Obvious Over Bolosky

The Examiner rejected claims 81, 89, 96, 100, 105, 107 and 115 under 35 U.S.C. § 103(a) as being obvious over Bolosky (described in Section VIII.A.1 above). A prima facie case of obviousness for rejecting these claims has not been established. The cited reference does not disclose or suggest Applicant's claimed invention. The Patent and Trademark Office's burden of establishing a prima facie case of obviousness is not met unless "'the teachings from the prior art itself would appear to have suggested the claimed subject matter to a person of ordinary skill in the art.'" In re Bell, 26 U.S.P.Q. 2d 1529, 1531 (Fed. Cir. 1993)(quoting In re Rinehart, 189 U.S.P.Q. 143,147 (C.C.P.A. 1976)).

1. Bolosky Does Not Disclose or Suggest the Scalability Aspect of the Claimed Invention

Dependent claims 81, 89, 96, 105 and 107 are directed to the scalability aspect of the claimed invention. Claim 81 depends from independent claim 79; claim 89 depends from

independent claim 82; claim 96 depends from independent claim 91; claim 105 depends from independent claim 104; and claim 107 depends from independent claim 106. Each of these independent claims require that a server operates independently of a data storage device so as to permit the addition (or removal) of a server without the addition (or removal) of a data storage device. Bolosky does not disclose or suggest this limitation. Rather, as discussed in Section VIII.A.2 above, Bolosky discloses multiple subsystems that each include a microprocessor tied to one or more data storage devices. Thus, because the Examiner has failed to meet his burden of establishing a prima facie case of obviousness, claims 81, 89, 96, 105 and 107 should be allowed.

2. Bolosky Does Not Disclose or Suggest the Survivability Aspect of the Claimed Invention

Dependent claims 100 and 115 are directed to the survivability aspect of the claimed invention. Claim 100 depends from independent claim 98; and claim 115 depends from independent claim 114. Each of these independent claims require first and second (or a plurality of) data storage devices that each store substantially the same data such that, in the event of a failure of any one of the data storage devices, the data is accessible from any other of the data storage devices that are operable. Bolosky does not disclose or suggest this limitation. Rather, as discussed in Section VIII.A.3 above, Bolosky discloses data storage devices that each store different blocks of data such that no one data storage device stores substantially the same data as any other data storage device. Thus, because the Examiner has failed to meet his burden of establishing a prima facie case of obviousness, claims 100 and 115 should be allowed.

3. Different Groups of Claims are Separately Patentable

Claims 81, 89, 96, 105 and 107 are directed to the scalability aspect of the claimed invention, and are not obvious over Bolosky for the reasons discussed in Section VIII.B.1 above. Claims 100 and 115 are directed to the survivability aspect of the claimed invention, and are not obvious over Bolosky for the reasons discussed in Section VIII.B.2 above.

C. Applicant's Claims are not Obvious Over Bolosky in View of Peacock

The Examiner rejected claims 90, 97, 99 and 101-103 under 35 U.S.C. § 103(a) as being obvious over Bolosky (described in Section VIII.A.1 above) in view of U.S. Patent No. 4,914,570 to Peacock ("Peacock"), attached hereto as Appendix D. Peacock discloses a multiple processor computer system. A prima facie case of obviousness for rejecting these claims has not been established. The cited references do not disclose or suggest Applicant's claimed invention. Furthermore, these cited references are not properly combinable. Still further, even if these cited reference are combined, they do not disclose or suggest Applicant's claimed invention. The Patent and Trademark Office's burden of establishing a prima facie case of obviousness is not met unless "'the teachings from the prior art itself would appear to have suggested the claimed subject matter to a person of ordinary skill in the art.'" In re Bell, 26 U.S.P.Q. 2d 1529, 1531 (Fed. Cir. 1993)(quoting In re Rinehart, 189 U.S.P.Q. 143,147 (C.C.P.A. 1976)).

1. Bolosky and Peacock Do Not Disclose or Suggest the Scalability Aspect of the Claimed Invention and are not Properly Combinable

Independent claim 102 and dependent claims 90, 97 and 103 are directed to the scalability aspect of the claimed invention.[3] Claim 90 depends from independent claim 82; claim

[3] Dependent claim 103 is also directed to the survivability aspect of the claimed invention, discussed in Section VIII.C.2 below.

97 depends from independent claim 91; and claim 103 depends from independent claim 102. Independent claim 102 and independent claims 82 and 91 each require that a server operates independently of a data storage device so as to permit the addition (or removal) of a server without the addition (or removal) of a data storage device. Neither Bolosky nor Peacock disclose or suggest this limitation. Rather, as discussed in Section VIII.A.2 above, Bolosky discloses multiple subsystems that each include a microprocessor tied to one or more data storage devices. Peacock merely discloses a multiple processor computer system in which each of the processors has its own associated memory, but also has access to the memories of the other processors. Thus, Applicant's claimed invention is clearly distinguishable from Bolosky and Peacock.

Furthermore, "[b]efore the PTO may combine the disclosures of two or more prior art references in order to establish prima facie obviousness, there must be some suggestion for doing so, found either in the references themselves or in the knowledge generally available to one of ordinary skill in the art." In re Jones, 21 U.S.P.Q. 2d 1941, 1943-44 (Fed. Cir. 1992). If there is no technological motivation for modifying a reference, then the reference should not be part of a §103 rejection.

There is no motivation to combine Bolosky and Peacock. Bolosky discloses the various components of a media server system, such as a video-on-demand system. By contrast, Peacock discloses the inner-workings of a multiple processor computer system. Nothing in either reference suggests that any one of the various components of Bolosky could be modified in accordance with the teachings of Peacock.

Still further, even if they are combined, the combination of Bolosky and Peacock does not disclose or suggest Applicant's claimed invention. Specifically, the combination does not disclose or suggest a server that operates independently of a data storage device so as to

permit the addition (or removal) of a server without the addition (or removal) of a data storage device., as claimed by Applicant.

Thus, because the Examiner has failed to meet his burden of establishing a prima facie case of obviousness, claims 90, 97, 102 and 103 should be allowed.

2. Bolosky and Peacock Do Not Disclose or Suggest the Survivability Aspect of the Claimed Invention and are not Properly Combinable

Dependent claims 99, 101 and 103 are directed to the survivability aspect of the claimed invention.[4] Claims 99 and 101 depend from independent claim 98; and claim 103 depends from independent claim 102. Independent claims 98 and 102 each require a plurality of data storage devices that each store substantially the same data such that, in the event of a failure of any one of the data storage devices, the data is accessible from any other of the data storage devices that are operable. Neither Bolosky nor Peacock disclose or suggest this limitation. Rather, as discussed in Section VIII.A.3 above, Bolosky discloses data storage devices that each store different blocks of data such that no one data storage device stores substantially the same data as any other data storage device. Peacock merely discloses a multiple processor computer system in which each of the processors has its own associated memory. Thus, Applicant's claimed invention is clearly distinguishable from Bolosky and Peacock.

Furthermore, "[b]efore the PTO may combine the disclosures of two or more prior art references in order to establish prima facie obviousness, there must be some suggestion for doing so, found either in the references themselves or in the knowledge generally available to one of ordinary skill in the art." In re Jones, 21 U.S.P.Q. 2d 1941, 1943-44 (Fed. Cir. 1992). If

[4] Dependent claim 103 is also directed to the scalability aspect of the claimed invention, discussed in Section VIII.C.1 above.

there is no technological motivation for modifying a reference, then the reference should not be part of a §103 rejection.

There is no motivation to combine Bolosky and Peacock. Bolosky discloses the various components of a media server system, such as a video-on-demand system. By contrast, Peacock discloses the inner-workings of a multiple processor computer system. Nothing in either reference suggests that any one of the various components of Bolosky could be modified in accordance with the teachings of Peacock.

Still further, even if they are combined, the combination of Bolosky and Peacock does not disclose or suggest Applicant's claimed invention. Specifically, the combination does not disclose or suggest a plurality of data storage devices that each store substantially the same data such that, in the event of a failure of any one of the data storage devices, the data is accessible from any other of the data storage devices that are operable, as claimed by Applicant.

Thus, because the Examiner has failed to meet his burden of establishing a prima facie case of obviousness, claims 99, 101 and 103 should be allowed.

3. Different Groups of Claims are Separately Patentable

Claims 90, 97 and 102 are directed to the scalability aspect of the claimed invention, and are not obvious over Bolosky in view of Peacock for the reasons discussed in Section VIII.C.1 above. Claims 99 and 101 are directed to the survivability aspect of the claimed invention, and are not obvious over Bolosky in view of Peacock for the reasons discussed in Section VIII.C.2 above. Claim 103 is directed to both the scalability and survivability aspects of the claimed invention, and is not obvious over Bolosky in view of Peacock for the reasons discussed in Sections VIII.C.1 and VIII.C.2 above.

IX. APPENDICES

Attached hereto are the following Appendices:

Appendix A – Claims on Appeal

Appendix B – Figs. 1 and 2 of the Application

Appendix C – U.S. Patent No. 5,699,503 to Bolosky et al.

Appendix D – U.S. Patent No. 4,914,570 to Peacock

X. SUMMARY

For the foregoing reasons, Applicant respectfully submits that claims 79-115 are patentable over the cited references and should be allowed. Accordingly, Applicant respectfully requests that the Board reverse the Examiner's rejections of claims 79-115, and allow claims 79-115.

Respectfully submitted,

By: Judith L. Carlson

Judith L. Carlson, Reg. No. 41,904
STINSON MORRISON HECKER LLP
1201 Walnut Street, Suite 2800
P.O. Box 419251
Kansas City, MO 64141-6251
Telephone: (816) 842-8600
Facsimile: (816) 691-3495
Attorney for Applicant

A

THIS PAGE BLANK (USPTO)

APPENDIX A

Claims on Appeal

The text of the claims on appeal are as follows:

79. A scalable system for providing network processing and stored data access, the system comprising:

(a) a server operative to process user requests;

(b) a switch operatively connected to the server;

(c) a data storage device operatively connected to the switch; and

(d) wherein the server operates independently of the data storage device and is connected to the data storage device via the switch in a manner to permit the inclusion of an additional server to process other user requests without the inclusion of an additional data storage device.

80. The system of Claim 79 wherein the server operates independently of the data storage device and is connected to the data storage device via the switch in a manner to permit the inclusion of an additional data storage device without the inclusion of an additional server.

81. The system of Claim 79 wherein the server applies an application to the user requests, the application selected from the group consisting of: a mail application, a news application, a directory application, a content application, a groupware application, and an internet protocol (IP) service.

82. A scalable system for providing network processing and stored data access, the system comprising:

(a) at least first and second servers operative to process at least first and second user requests, respectively;

(b) a switch operatively connected to each of the servers;

(c) a plurality of data storage devices operatively connected to the switch; and

(d) wherein the servers operate independently of the data storage devices and are connected to the data storage devices via the switch in a manner to permit the inclusion of an additional server to process at least an additional user request without the inclusion of an additional data storage device.

83. The system of Claim 82 wherein the servers operate independently of the data storage devices and are connected to the data storage devices via the switch in a manner to permit the inclusion of an additional data storage device without the inclusion of an additional server.

84. The system of Claim 83 wherein the servers operate independently of the data storage devices and are connected to the data storage devices via the switch in a manner to permit the removal of any one of the plurality of data storage devices without the removal of any of the servers.

85. The system of Claim 82 wherein the servers operate independently of the data storage devices and are connected to the data storage devices via the switch in a manner to permit the removal of any one of the servers without the removal of any of the data storage devices.

86. The system of Claim 82 wherein each of the first and second servers applies an application, the application applied by the first server being substantially the same as the application applied by the second server such that, in the event of a failure of either of the first and second servers, any subsequent user requests will be processed by any other of the servers that are operable.

87. The system of Claim 82 wherein each of the plurality of data storage devices stores data, the data stored by each of the plurality of data storage devices being substantially the same such that, in the event of a failure of any one of the plurality of data storage devices, the data is accessible from any other of the plurality of data storage devices that are operable.

88. The system of Claim 82 wherein each of the first and second servers applies an application, the application applied by the first server being substantially the same as the application applied by the second server such that, in the event of a failure of either of the first and second servers, any subsequent user requests will be processed by any other of the servers that are operable, and wherein each of the plurality of data storage devices stores data, the data stored by each of the plurality of data storage devices being substantially the same such that, in

the event of a failure of any one of the plurality of data storage devices, the data is accessible from any other of the plurality of data storage devices that are operable.

89. The system of Claim 82 wherein each of the at least first and second servers applies an application selected from the group consisting of: a mail application, a news application, a directory application, a content application, a groupware application, and an internet protocol (IP) service.

90. The system of Claim 82 further comprising a load balancer operatively connected to each of the at least first and second servers, the load balancer operative to route an additional user request to the one of the at least first and second servers with the least load.

91. A scalable system for providing network processing and stored data access, the system comprising:

(a) at least first and second sets of servers, each of the sets of servers comprising at least first and second servers operative to process at least first and second user requests, respectively, and wherein each of the sets of servers applies a separate application;

(b) a switch operatively connected to each of the servers within each of the sets of servers;

(c) a plurality of data storage devices operatively connected to the switch; and

(d) wherein the sets of servers operate independently of the data storage devices and are connected to the data storage devices via the switch in a manner to permit the

inclusion of an additional server to any of the sets of servers to process at least an additional user request without the inclusion of an additional data storage device.

92. The system of Claim 91 wherein the sets of servers operate independently of the data storage devices and are connected to the data storage devices via the switch in a manner to permit the inclusion of an additional data storage device without the inclusion of an additional server to any of the sets of servers.

93. The system of Claim 92 wherein the sets of servers operate independently of the data storage devices and are connected to the data storage devices via the switch in a manner to permit the removal of any one of the plurality of data storage devices without the removal of any of the servers from any of the sets of servers.

94. The system of Claim 93 wherein the data stored by any one of the plurality of data storage devices is associated with an application applied by any one of the sets of servers.

95. The system of Claim 91 wherein the sets of servers operate independently of the data storage devices and are connected to the data storage devices via the switch in a manner to permit the removal of any one of the servers from any one of the sets of servers without the removal of any of the plurality of data storage devices.

96. The system of Claim 91 wherein each of the at least first and second servers of any one of the sets of servers applies an application selected from the group consisting of: a mail application, a news application, a directory application, a content application, a groupware application, and an internet protocol (IP) service.

97. The system of Claim 91 wherein each of the at least first and second servers of any one of the sets of servers applies an application, and wherein the system further comprises a load balancer operatively connected to each of the at least first and second servers of each of the sets of servers, the load balancer operative to route user requests to the one of the at least first and second servers of the sets of servers with the least load for a particular application.

98. A survivable system for providing network processing and stored data access, the system comprising:

(a) at least first and second servers operative to process at least first and second user requests, respectively,

(b) a switch operatively connected to each of the servers;

(c) a plurality of data storage devices operatively connected to the switch;

(d) wherein each of the first and second servers applies an application, the application applied by the first server being substantially the same as the application applied by the second server such that, in the event of a failure of either of the first and second servers, any subsequent user requests will be processed by any other of the servers that are operable; and

(e) wherein each of the plurality of data storage devices stores data, the data stored by each of the plurality of data storage devices being substantially the same such that, in the event of a failure of any one of the plurality of data storage devices, the data is accessible from any other of the plurality of data storage devices that are operable.

99. The system of Claim 98 wherein the data stored by any one of the plurality of data storage devices is associated with an application applied by any one of the first and second servers.

100. The system of Claim 98 wherein each of the at least first and second servers applies an application selected from the group consisting of: a mail application, a news application, a directory application, a content application, a groupware application, and an internet protocol (IP) service.

101. The system of Claim 98 further comprising a load balancer operatively connected to each of the at least first and second servers, the load balancer operative to route user requests to the one of the at least first and second servers corresponding to the server with the least load.

102. A scalable system for providing network processing and stored data access, the system comprising:

(a) at least first and second servers operative to process at least first and second user requests, respectively;

(b) a switch operatively connected to each of the servers;

(c) a plurality of data storage devices operatively connected to the switch;

(d) a load balancer operatively connected to each of the at least first and second servers, the load balancer operative to route user requests to the one of the at least first and second servers with the least load; and

(e) wherein the servers operate independently of the data storage devices and are connected to the data storage devices via the switch in a manner to permit the inclusion of an additional server to process at least an additional user request without the inclusion of an additional data storage device, to permit the inclusion of an additional data storage device without the inclusion of an additional server, to permit the removal of any one of the servers without the removal of any of the data storage devices, and to permit the removal of any one of the data storage devices without the removal of any of the servers.

103. The system of Claim 102 wherein each of the first and second servers applies an application, the application applied by the first server being substantially the same as the application applied by the second server such that, in the event of a failure of either of the first and second servers, any subsequent user requests will be processed by any other of the servers that are operable, and wherein each of the plurality of data storage devices stores data, the data stored by each of the plurality of data storage devices being substantially the same such that, in the event of a failure of any one of the plurality of data storage devices, the data is accessible from any other of the plurality of data storage devices that are operable.

104. A method for providing network processing and stored data access, the method comprising the steps of:

(a) providing a server operative to apply an application;

(b) receiving a user request on the server;

(c) applying the application to the user request to generate a query;

(d) providing a data storage device configured to store data;

(e) switching the query to the data storage device;

(f) routing requested data from the data storage device to the server in response to the query; and

(g) providing an additional server without providing an additional data storage device, or alternatively, providing an additional data storage device without providing an additional server.

105. The method of Claim 104 wherein the application is selected from the group consisting of: a mail application, a news application, a directory application, a content application, a groupware application, and an internet protocol (IP) service.

106. A method for providing network processing and stored data access, the method comprising the steps of:

(a) providing at least first and second servers operative to apply first and second applications, respectively;

(b) receiving first and second user requests on the first and second servers, respectively;

(c) applying the first and second applications to the first and second user requests, respectively, to generate first and second queries, respectively;

(d) providing at least first and second data storage devices configured to store first and second data, respectively;

(e) switching the first and second queries to the first and second data storage devices, respectively;

(f) routing first requested data from the first data storage device to the first server in response to the first query, and routing second requested data from the second data storage device to the second server in response to the second query; and

(g) providing an additional server without providing an additional data storage device, or alternatively, providing an additional data storage device without providing an additional server.

107. The method of Claim 106 wherein each of the first and second applications is selected from the group consisting of: a mail application, a news application, a directory application, a content application, a groupware application, and an internet protocol (IP) service.

108. The method of Claim 106 wherein the first application is substantially the same as the second application.

109. The method of Claim 108 further comprising the step of:

(h) in the event of a failure of either of the first and second servers, processing any subsequent user requests on any other of the servers that are operable.

110. The method of Claim 106 wherein the first data is substantially the same as the second data.

111. The method of Claim 110 further comprising the step of:

(h) in the event of a failure of either of the first and second data storage devices, providing any subsequent requested data from any other of the data storage devices that are operable.

112. The method of Claim 106 wherein the first application is substantially the same as the second application, and wherein the first data is substantially the same as the second data.

113. The method of Claim 112 further comprising the steps of:

(h) in the event of a failure of either of the first and second servers, processing subsequent requests on any other of the servers that are operable; and

(i) in the event of a failure of either of the first and second data storage devices, providing any subsequent requested data from any other of the data storage devices that are operable.

114. A method for providing network processing and stored data access, the method comprising the steps of:

(a) providing at least first and second servers operative to apply first and second applications, respectively, the first application being substantially the same as the second application;

(b) receiving first and second user requests on the first and second servers, respectively;

(c) applying the first and second applications to the first and second user requests, respectively, to generate first and second queries, respectively;

(d) providing at least first and second data storage devices configured to store first and second data, respectively, the first data being substantially the same as the second data;

(e) switching the first and second queries to the first and second data storage devices, respectively;

(f) routing first requested data from the first data storage device to the first server in response to the first query, and routing second requested data from the second data storage device to the second server in response to the second query;

(g) in the event of a failure of either of the first and second servers, processing any subsequent requests on any other of the servers that are operable; and

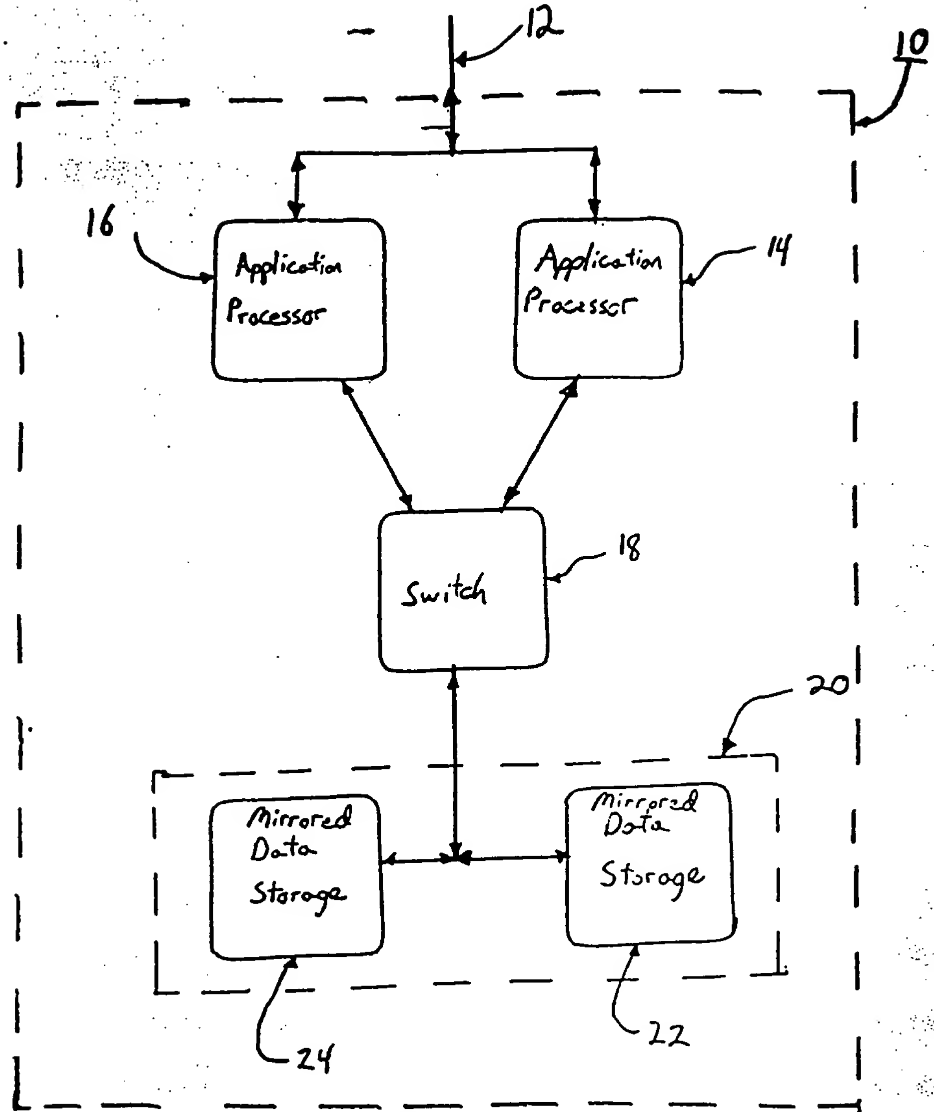
(h) in the event of a failure of either of the first and second data storage devices, providing any subsequent requested data from any other of the data storage devices that are operable.

115. The method of Claim 114 wherein each of the first and second applications is selected from the group consisting of: a mail application, a news application, a directory application, a content application, a groupware application, and an internet protocol (IP) service.

B

APPENDIX B

Figs. 1 and 2 of the Application


12
10
16
Application Processor
Application Processor
14
Switch
18
20
Mirrored Data Storage
Mirrored Data Storage
24
22
FIG. 1

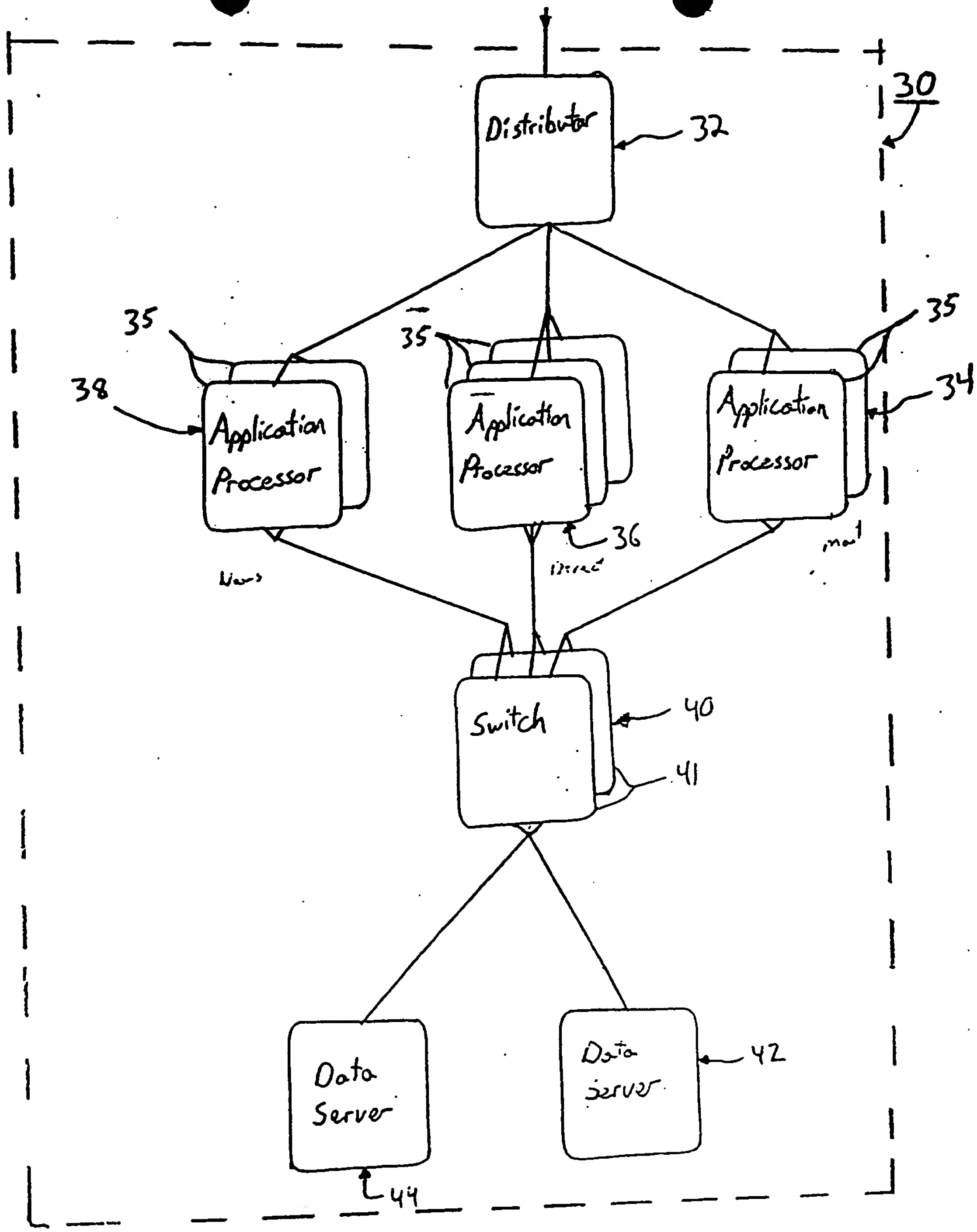

30
Distributor
32
35
35
35
38
Application Processor
Application Processor
Application Processor
34
36
Switch
40
41
Data Server
Data Server
42
44

FIG. 2

APPENDIX C

U.S. Patent No. 5,699,503 to Bolosky et al.

US005699503A

United States Patent [19]

Bolosky et al.

[11] Patent Number: 5,699,503

[45] Date of Patent: Dec. 16, 1997

[54] METHOD AND SYSTEM FOR PROVIDING FAULT TOLERANCE TO A CONTINUOUS MEDIA SERVER SYSTEM

[75] Inventors: William J. Bolosky, Issaquah; Robert P. Fitzgerald, Redmond; Lawrence W. Osterman, Woodinville, all of Wash.

[73] Assignee: Microsoft Corporation, Redmond, Wash.

[21] Appl. No.: 705,075

[22] Filed: Aug. 26, 1996

Related U.S. Application Data

[63] Continuation of Ser. No. 437,935, May 9, 1995, abandoned.

[51] Int. Cl.[6] G06F 11/00

[52] U.S. Cl. 395/182.04

[58] Field of Search 395/182.02, 182.03, 395/182.04, 182.06, 182.05; 348/7, 12, 11, 13, 5; 455/4.2, 5.1

[56] References Cited

U.S. PATENT DOCUMENTS

3,771,143	11/1973	Taylor.	
4,959,774	9/1990	Davis	395/575
5,271,012	12/1993	Blaum et al.	371/10.1
5,333,305	7/1994	Neufeld	395/575
5,357,509	10/1994	Ohizumi	371/10.1
5,392,244	2/1995	Jacobson et al.	365/200
5,414,455	5/1995	Hooper et al.	348/7
5,422,674	6/1995	Hooper et al.	348/409
5,440,336	8/1995	Buhro et al.	348/13
5,442,390	8/1995	Hooper et al.	348/7
5,473,362	12/1995	Fitzgerald et al.	348/7
5,559,764	9/1996	Chen et al.	369/30
5,606,359	2/1997	Youden et al.	348/7

OTHER PUBLICATIONS

Chen et al., RAID: High-Performance, Reliable Secondary Storage, ACM Computing Surveys, Jun. 1994, at 145.

Reddy & Banerjee, An Evaluation of Multiple-Disk I/O Systems, IEEE Transactions on Computers, Dec. 1989, ay 1680.

Nexis search (of articles listing the announcement of Microsoft's Tiger media server. Search performed Aug. 17, 1995.

So?*, Driving Microsoft, InformationWEEK, May 1994, at 100.

Derwent Research Disclosure RD 345097 (Anonymous), Dual Striping Method for Replicated Data Disc Array, Jan. 10, 1993.

Hsiao & DeWitt, A persofrmance Study of Three High Availabiltiy Data Replication Strategies, 1991 Parallel and Distributed Information Systems Intl. Conf., at 18.

Golubchik et al., Chained Declustering: Load Balancing and Rubustness to Skew and Failures, Research Issues in Data Engineering 1992 Workshop, at 88.

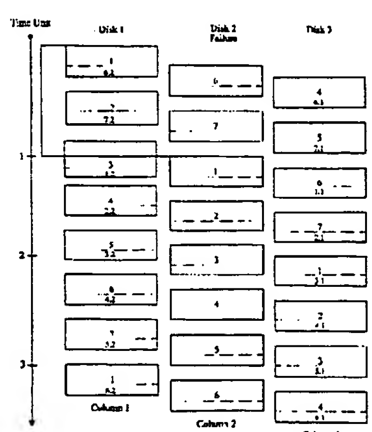
(List continued on next page.)

Primary Examiner—Robert W. Beausoliel, Jr.
Assistant Examiner—Norman M. Wright
Attorney, Agent, or Firm—Seed and Berry LLP

[57] ABSTRACT

A method and system is provided for tolerating component failure in a continuous media server system. The present invention guarantees data streams at a constant rate to subscribers for the data streams even when at least one component fails. The present invention is able to guarantee data streams at a constant rate by utilizing declustered mirroring and by reserving bandwidth for both normal mode processing and failure mode processing. The declustered mirroring of the present invention is performed by dividing the data to be stored in the continuous media server system into blocks. The blocks are then striped across the storage devices of the continuous media server system and each block is divided into a number of sub-blocks. The sub-blocks are in turn stored on separate storage devices. The present invention reserves bandwidth for both normal mode processing and failure mode processing. Since the present invention utilizes declustered mirroring, the bandwidth reserved for failure mode processing is reduced. Therefore, when a failure occurs, the bandwidth reserved for failure mode processing is utilized and the data streams to the subscribers are uninterrupted.

23 Claims, 10 Drawing Sheets



OTHER PUBLICATIONS

Catania et al., Performance Evaluation of a partial Dynamic Declustering Disk Array System, 1994 High Performance Distributed Computing Intl. Symp., at 224.

Mace, Oracle Media Server Widely Endorsed, InfoWorld, Feb. 21, 1994, at 20.

Li et al., Combining Replication and Parity Approaches for Fault-Tolerant Disk Arrays, 1994 Parallel & Distributed Processing Symposium, at 360.

Buck, The Oracle Media Server for nCUBE Massively Parallel Systems, 1994 Parallel Processing Symposium, at 670.

Vina et al., Real-Time Multimedia Systems, 13th IEEE Symposium on Mass Storage (1994), at 77.

Kovalick, The Video Server as a Component in Interactive Broadband Delivery Systems, 1994 Community Networking Intergrated Multimedia Service Wrkshp. at 77.

Kenchammana-Hosekote & Srivastava, Scheduling Continious Media in a Video-On-Demand Server, 1994 International Multimedia Conference, at 19.

Gibson, Garth A., et al., *Failure Correction Techniques for Large Disk Arrays*, APLOS III, Apr. 1989, Boston, MA, pp. 123-132.

Patterson, David A. et al., *A Case for Redundant Arrays of Inexpensive Disks (RAID), ACM SIGMOD Conference Proceedings*, Chicago, IL, Jun. 1-3, 1988, pp. 109-116.

Bitton, Dina and Jim Gray, *Disk Shadowing*, Proceedings of the 14th VLDB Conference, Los Angeles, CA 1988, pp. 331-338.

Chen, Peter M., et al., *An Evaluation of Redundant Arrays of Disks using an Amdahl 5890*, Proceedings of the ACM SIGMETRICS Conference, 1990, pp. 74-85.

Copeland, George and Tom Keller, *A Comparison of High-Availability Media Recovery Techniques*, Proceedings of the ACM SIGMOD Conference, 1989, pp. 98-109.

Gray, Jim et al., *Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput*, Proceedings of the 16th Conference on Very Large Data Bases, Brisbane, AU, 1990, pp. 148-161.

Muntz, Richard R. and John C.S. Lui, *Performance Analysis of Disk Arrays Under Failure*, Proceedings of the 16th VLDB Conference, Brisbane, AU, 1990, pp. 162-173.

Chen, Peter M. and David A. Patterson, *Maximizing Performance in a Striped Disk Array*, Proceedings of the ACM SIGARCH Conference, 1990, pp. 322-331.

How RAID Works, Computerworld, Mar. 14, 1994, v28, n11, pp. 92-97.

Hsiao, Hui-I and David J. DeWitt, *Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines*, Proceedings of the 6th International Data Engineering Conference, Los Angeles, CA, 1990, pp. 456-465.

Kliewer, Bradley Dyck, *Raid To The Rescue*, Varbusiness, Fall 1992, pp. 39-42.

Simpson, David, *Balancing RAID delivery*, Digital News & Reviews, Dec. 20, 1993, pp. 43-48.

Marson, Carolyn Duffy, *RAID—New storage devices take feds by storm*, Federal Computer Week, Sep. 14, 1992, pp. 16-18.

McBride, John, *Getting the most out of redundant arrays of inexpensive disks*, EDN, Feb. 4, 1993, pp. 109-114.

Bates, Ken, *RAID: a new balance of power*, DEC Professional, Jan. 1994, v13 n1 p34(5).

Pavlinik, Ed, *RAID: use as directed*, HP Professional, Aug. 1993, v7 n8 p30(4).

Holland, Mark and Garth A. Gibson, *Parity Declustering for Continuous Operation in Redundant Disk Arrays* ASPLOS V-10/92MA, USA.

Patterson, David A. et al., *Introduction to Redundant Arrays of Inexpensive Disks (RAID)*, Computer Science Division, Department of Electrical and Computer Science, University of California, Berkeley, CA, 1989.

Matloff, Norman S. and Raymond Wai-man Lo, *A "Greedy" Approach to the Write Problem in Shadowed Disk Systems*, Sixth International Conference on Data Engineering, Feb. 5-9, 1990, pp. 553-558.

The RAIDbook-A Source Book for Disk Array Technology, Raid Advisory Board, St. Peter, MN, Sep. 1, 1994, Chapters 1, 3, 4, 6 and 7.

Holland, Mark Calvin, *On-Line Data Reconstruction In Redundant Disk Arrays*, a dissertation submitted to the Department of Electrical and Computer Engineering, Carnegie Mellon University, in partial fulfillment of the requirements for the degree of Doctor Philosophy, 1994.

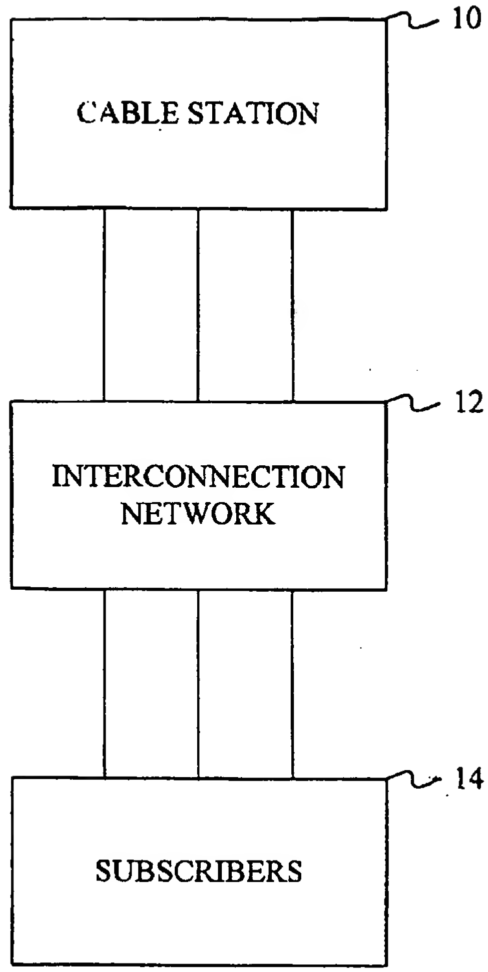

10
CABLE STATION
12
INTERCONNECTION NETWORK
14
SUBSCRIBERS

FIG. 1

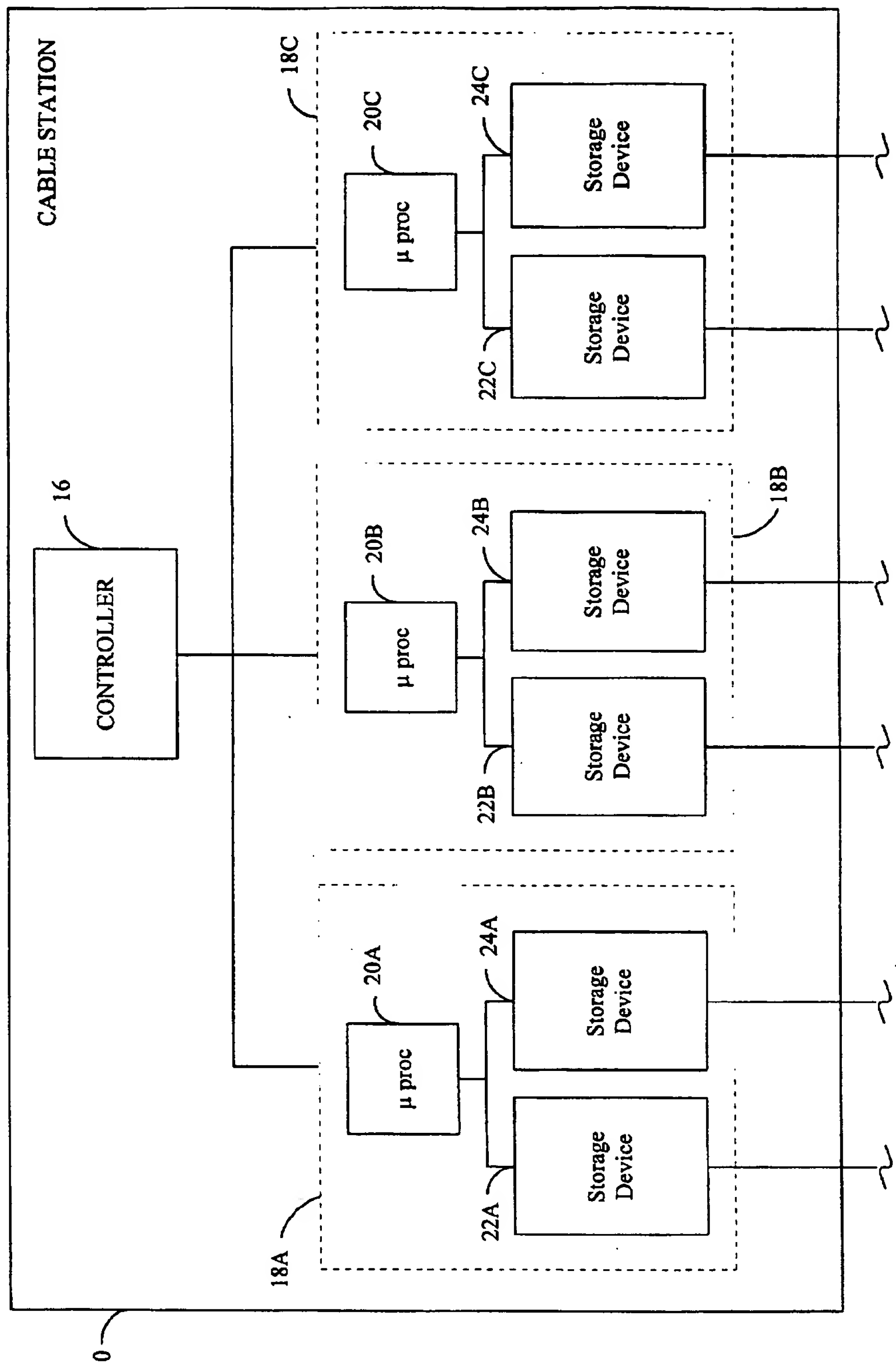

CABLE STATION
CONTROLLER
16
18A
18B
18C
20A
20B
20C
μ proc
μ proc
μ proc
22A
22B
22C
24A
24B
24C
Storage Device
Storage Device
Storage Device
Storage Device
Storage Device
Storage Device
10

FIG. 2

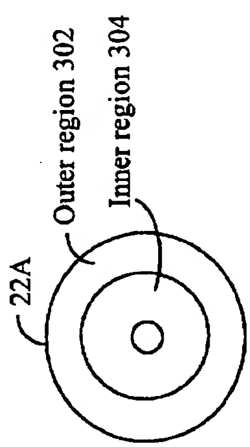


FIG. 3A

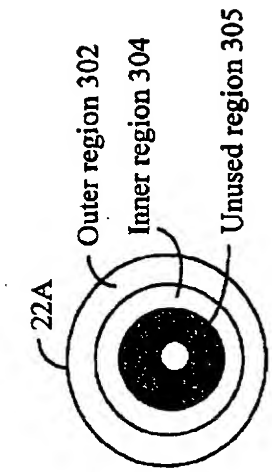


FIG. 3B

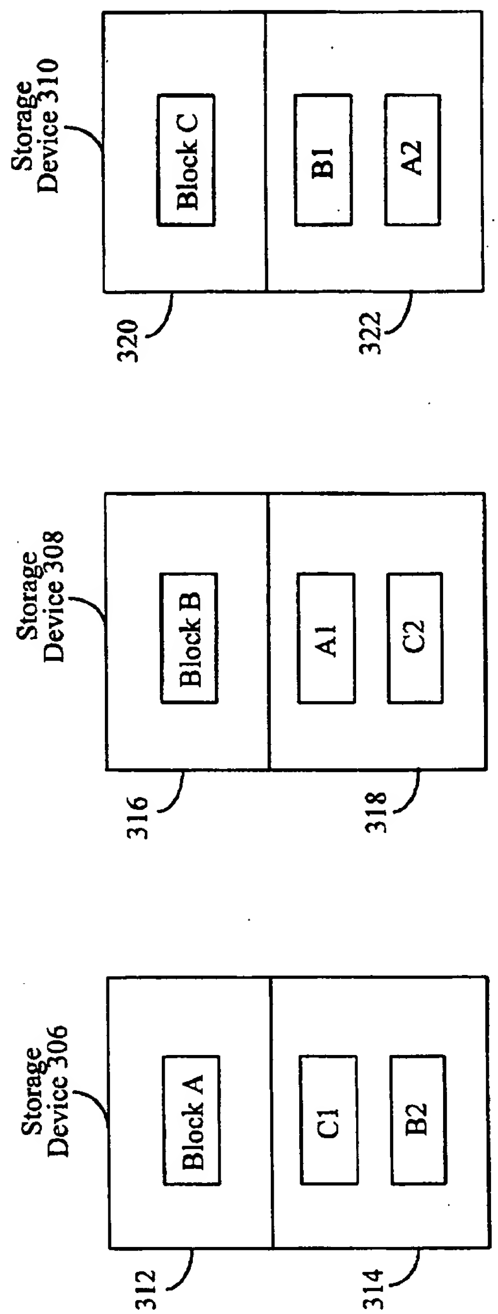


FIG. 3C

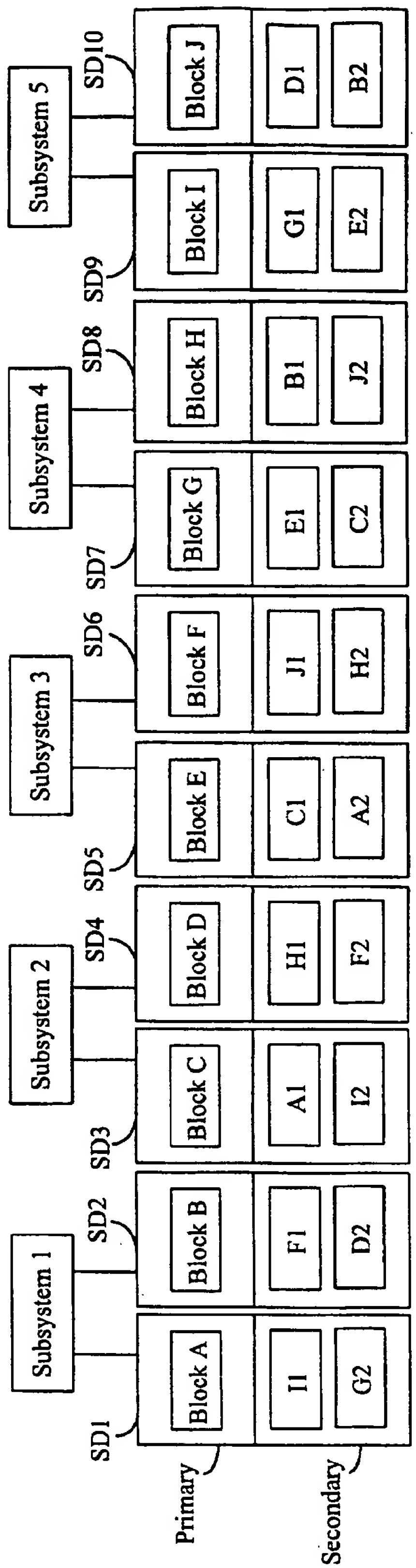
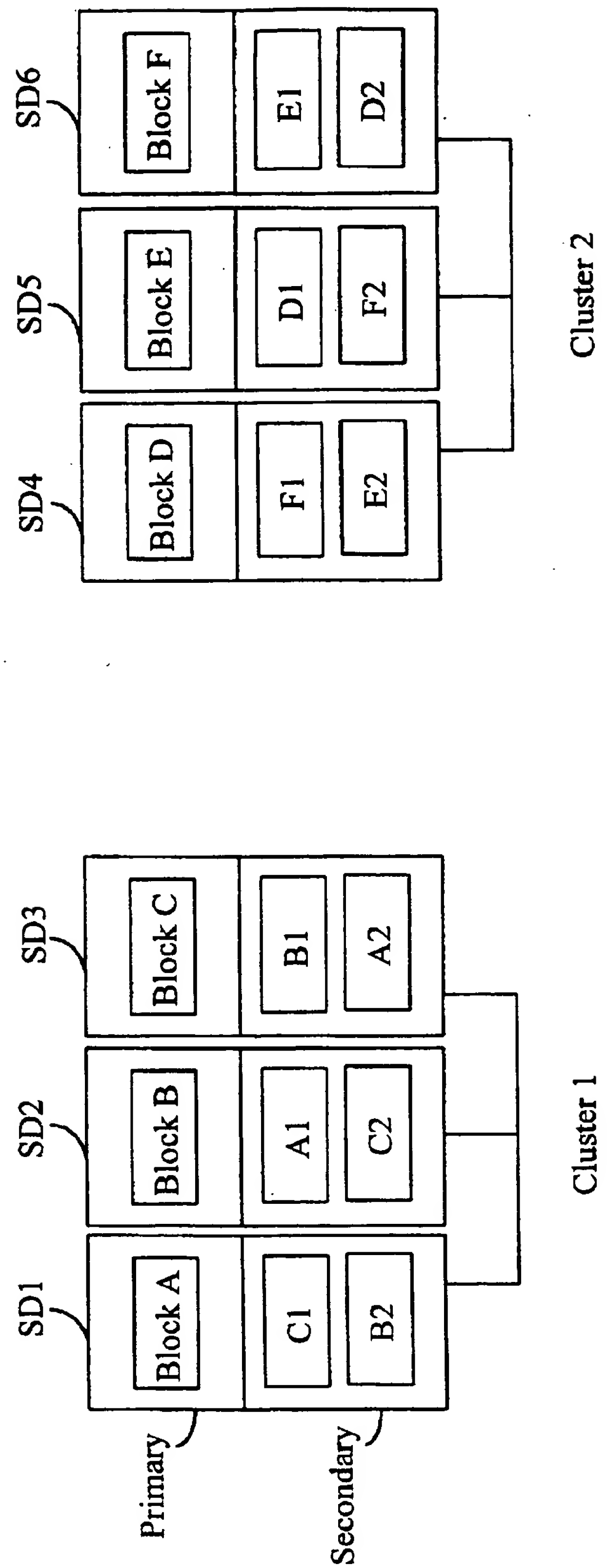

Subsystem 1
Subsystem 2
Subsystem 3
Subsystem 4
Subsystem 5
SD1
SD2
SD3
SD4
SD5
SD6
SD7
SD8
SD9
SD10
Block A
Block B
Block C
Block D
Block E
Block F
Block G
Block H
Block I
Block J
I1
G2
F1
D2
A1
I2
H1
F2
C1
A2
J1
H2
E1
C2
B1
J2
G1
E2
D1
B2
Primary
Secondary

FIG. 3D



SD1
Block A
C1
B2
SD2
Block B
A1
C2
SD3
Block C
B1
A2
Primary
Secondary
Cluster 1
SD4
Block D
F1
E2
SD5
Block E
D1
F2
SD6
Block F
E1
D2
Cluster 2

FIG. 3E

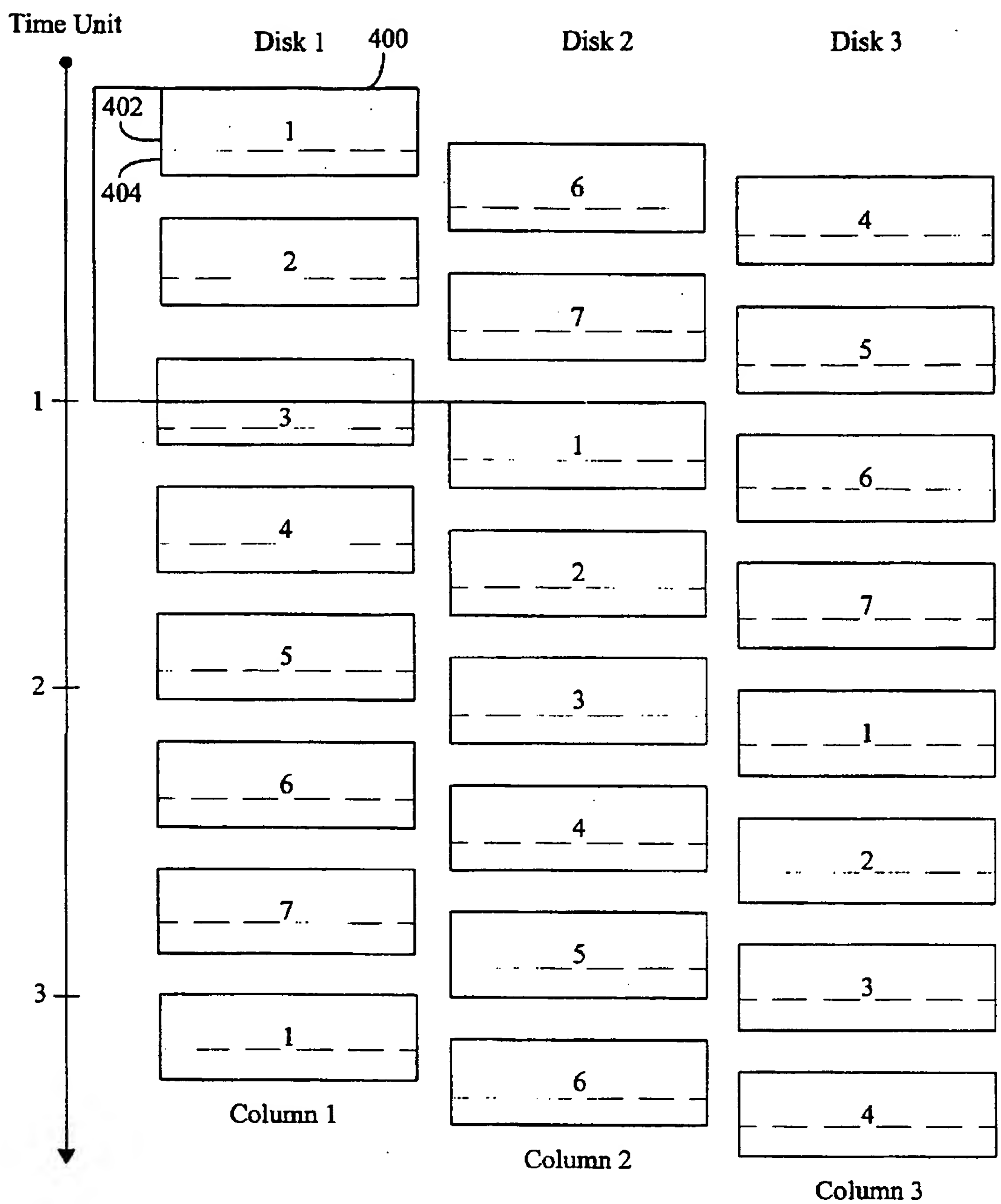

Time Unit
Disk 1
400
Disk 2
Disk 3
402
404
1
2
3
4
5
6
7
1
6
7
1
2
3
4
5
6
4
5
6
7
1
2
3
4
1
2
3
Column 1
Column 2
Column 3

FIG. 4

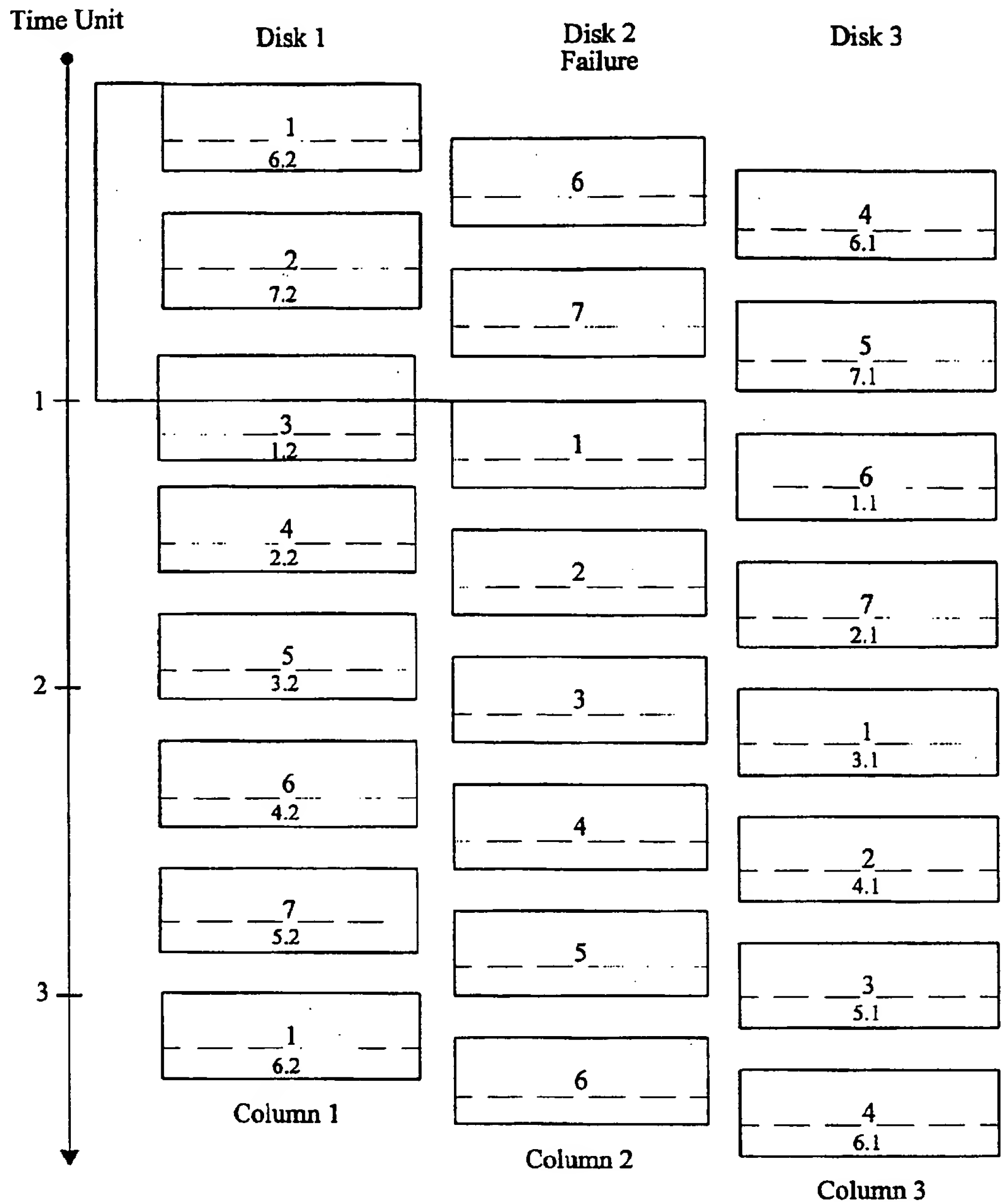

Time Unit
Disk 1
Disk 2
Failure
Disk 3
1
6.2
6
4
6.1
2
7.2
7
5
7.1
1
3
1.2
1
6
1.1
4
2.2
2
7
2.1
5
3.2
2
3
1
3.1
6
4.2
4
2
4.1
7
5.2
5
3
3
5.1
1
6.2
6
4
6.1
Column 1
Column 2
Column 3

FIG. 5

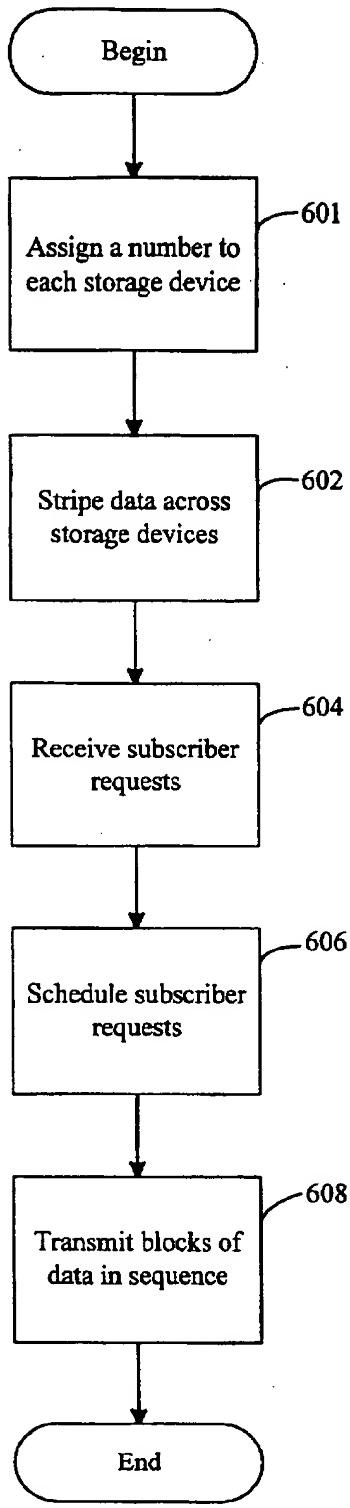

Begin
Assign a number to each storage device
601
Stripe data across storage devices
602
Receive subscriber requests
604
Schedule subscriber requests
606
Transmit blocks of data in sequence
608
End

FIG. 6

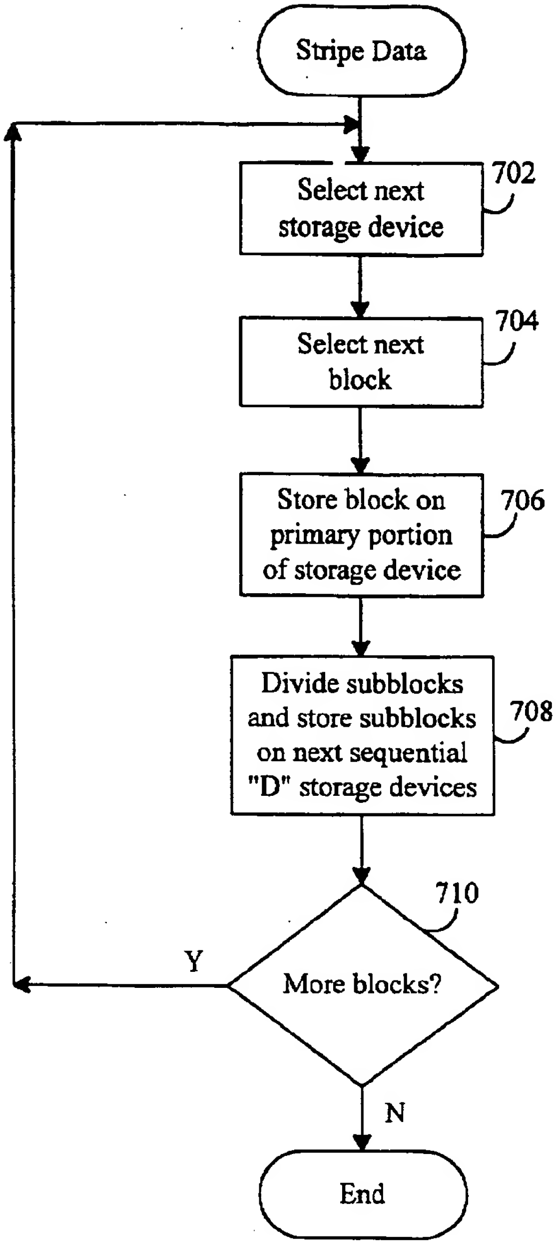

Stripe Data
Select next storage device
702
Select next block
704
Store block on primary portion of storage device
706
Divide subblocks and store subblocks on next sequential "D" storage devices
708
More blocks?
710
Y
N
End

FIG. 7

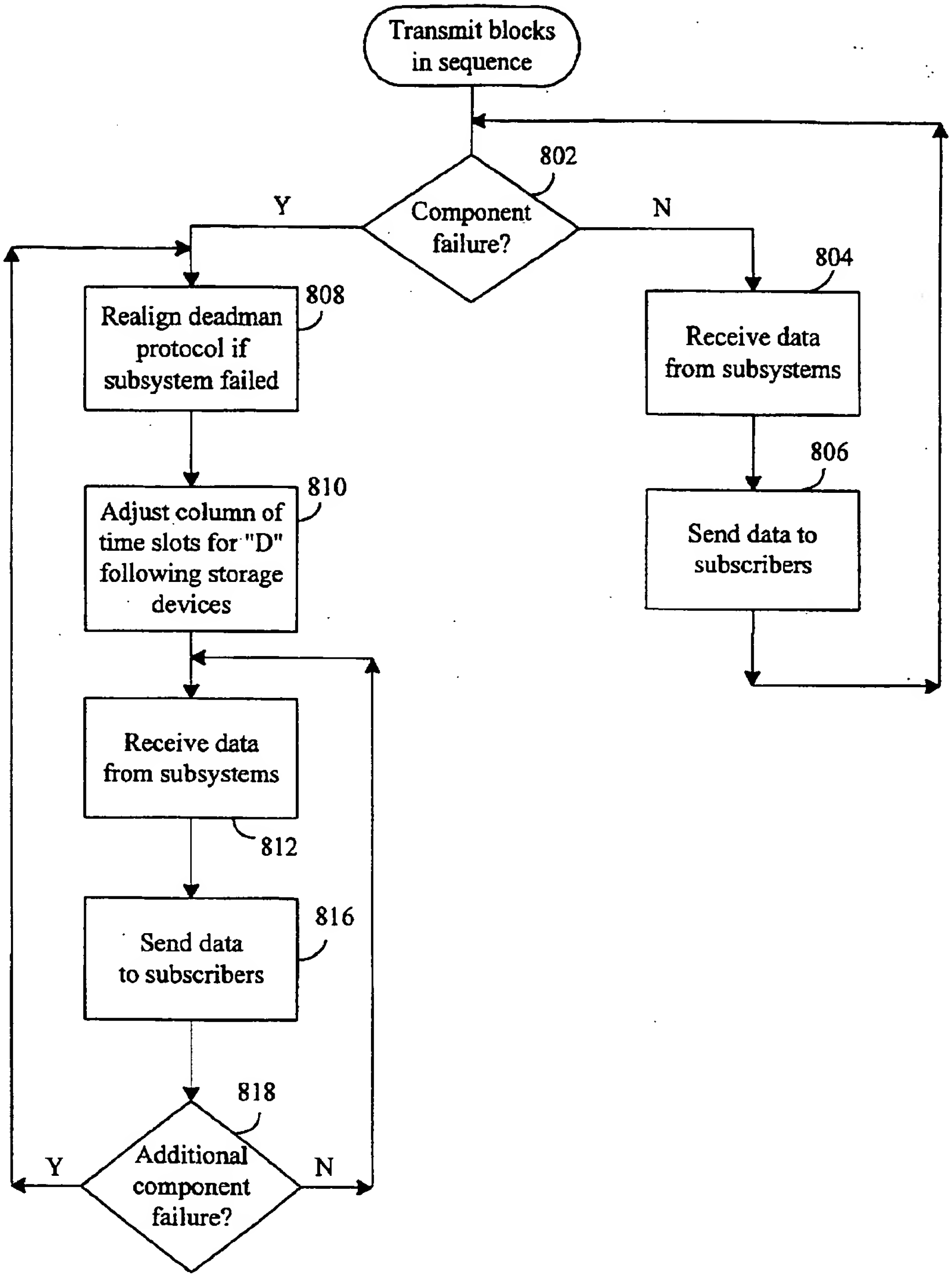

Transmit blocks in sequence
802
Component failure?
Y
N
804
Receive data from subsystems
806
Send data to subscribers
808
Realign deadman protocol if subsystem failed
810
Adjust column of time slots for "D" following storage devices
Receive data from subsystems
812
816
Send data to subscribers
818
Additional component failure?
Y
N

FIG. 8

METHOD AND SYSTEM FOR PROVIDING FAULT TOLERANCE TO A CONTINUOUS MEDIA SERVER SYSTEM

CROSS-REFERENCE TO RELATED APPLICATION

This application is a continuation of U.S. patent application Ser. No. 08/437,935, filed May 9, 1995, now abandoned.

TECHNICAL FIELD

The present invention relates generally to data processing systems and, more particularly, to fault tolerance in a continuous media server system.

BACKGROUND OF THE INVENTION

Some conventional data processing systems use a technique known as "mirroring" in order to continue operating when a storage device fails. Mirroring refers to a technique where for every storage device ("primary storage device") in a data processing system, the data processing system maintains a mirror storage device. The mirror storage device is a storage device that contains a duplicate copy of the data on the primary storage device. Whenever an operation is performed on the primary storage device that would alter the data contained thereon (e.g., write), the same operation is performed on the mirror storage device. Thus, at any given time, the mirror storage device has an exact duplicate copy of all the data on the primary storage device.

Since the mirror storage device has an exact duplicate copy of all the data on the primary storage device, if the primary storage device fails, the data processing system switches to use the mirror storage device and the operation of the data processing system continues with little interruption. Although mirroring provides for a more reliable data processing system, the mirroring technique is not suitable for all types of data processing systems since there must be a duplicate of every storage device on the system and since some interruption of the data processing system typically occurs.

One example of a data processing system where an interruption would not be acceptable, even for a short period of time, is a continuous media server system. A continuous media server system is a data processing system that typically has many storage devices and delivers data at a constant rate to subscribers for the data. In this context, the phrase "constant rate" refers to delivering the appropriate amount of data to a subscriber over a period of time, such as a second.

SUMMARY OF THE INVENTION

A method and system is provided for tolerating component failure in a continuous media server system. The present invention guarantees data streams at a constant rate to subscribers for the data streams even when at least one component fails. The present invention is able to guarantee data streams at a constant rate by utilizing declustered mirroring and by reserving bandwidth for both normal mode processing and failure mode processing. The declustered mirroring of the present invention is performed by dividing the data to be stored in the continuous media server system into blocks. The blocks are then striped across the storage devices of the continuous media server system and each block is divided into a number of sub-blocks. The sub-blocks are in turn stored on separate storage devices. The present invention reserves bandwidth for both normal mode processing and failure mode processing. Since the present invention utilizes declustered mirroring, the bandwidth reserved for failure mode processing is reduced. Therefore, when a failure occurs, the bandwidth reserved for failure mode processing is utilized and the data streams to the subscribers are uninterrupted.

In accordance with a first aspect of the present invention, a system is provided for delivering data to consumers at a constant rate. In accordance with this system of the first aspect of the present invention, the system comprises a plurality of sequentially numbered storage devices and a send component. The plurality of sequentially numbered storage devices contain data wherein the data comprises blocks and sub-blocks and the data is striped across the storage devices. A block is divided into a predefined number of sub-blocks and sub-blocks for a block on a first storage device are stored on the predefined number of storage devices that numerically follow the first storage device. The send component is for sending the blocks from the storage devices to the consumers and when a storage device fails, for sending the sub-blocks from the predefined number of storage devices that numerically follow the storage device that failed.

In accordance with a second aspect of the present invention, a method is provided in a continuous media system for delivering data to consumers at a constant rate. The continuous media system has a plurality of numerically sequential storage devices for storing data. The storage devices have a primary portion and a secondary portion and the data comprises numerically sequential blocks that are striped across the storage devices. In accordance with this method of the second aspect of the present invention, the blocks are stored on the primary portion of the storage devices such that after storing a block a next numerically sequential block is stored on a next numerically sequential storage device, the blocks are divided into a predefined number of sub-blocks and for each block, the sub-blocks for the block are stored on the secondary portion of the predefined number of storage devices that numerically follow a storage device on which the block is stored.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a video-on-demand system of a preferred embodiment of the present invention.

FIG. 2 is a more detailed block diagram of the cable station of FIG. 1.

FIG. 3A is a partial plan view of a storage device of FIG. 2 of the preferred embodiment of the present invention.

FIG. 3B is a partial plan view of a storage device of FIG. 2 of an alternative embodiment of the present invention.

FIG. 3C is a diagram depicting an example of storing data utilizing declustered mirroring on the storage devices of the preferred embodiment of the present invention.

FIG. 3D is a diagram depicting an example of storing data utilizing a first alternative embodiment of the present invention.

FIG. 3E is a diagram depicting an example of storing data utilizing a second alternative embodiment of the present invention.

FIG. 4 is a diagram illustrating the scheduling of bandwidth in a three disk drive system in accordance with the preferred embodiment of the present invention.

FIG. 5 is a diagram illustrating an example of the scheduling of bandwidth in the three disk drive system of FIG. 4

when a disk drive fails in accordance with the preferred embodiment of the present invention.

FIG. 6 depicts a high-level flow chart functionally illustrating the steps performed by the preferred embodiment of the present invention.

FIG. 7 depicts a flow chart of the steps performed by the preferred embodiment of the present invention for striping data across the storage devices.

FIG. 8 depicts a flow chart of the steps performed by the preferred embodiment of the present invention when transmitting data in normal mode processing and failure mode processing.

DETAILED DESCRIPTION OF THE INVENTION

The preferred embodiment of the present invention provides a method and system for tolerating component failure in a continuous media server system by utilizing declustered mirroring and by reserving bandwidth for failure mode processing. By utilizing the preferred embodiment of the present invention, subscribers to the continuous media server system are guaranteed a data stream at a constant rate even when one or more components of the continuous media server system fail. One example of a continuous media server system is a video-on-demand system where subscribers request video image sequences, such as movies, and the video-on-demand system guarantees a data stream of the video image sequences to the subscribers at a constant rate. In addition, the video-on-demand system can send a data stream of audio data to subscribers. In a video-on-demand system, it is important to guarantee data flow at a constant rate. Otherwise, when a subscriber is viewing a movie and a failure occurs, the movie will appear to have an interruption. Thus, the preferred embodiment can also be thought of as preventing data flow interruptions.

In the video-on-demand system of the present invention, the video-on-demand system has a number of storage devices where the data for the video image sequences is stored as blocks that are striped across the storage devices. The term "striped" refers to storing blocks sequentially across the storage devices and when the last storage device is reached, wrapping around and storing the next block on the first storage device. The data stream is sent to subscribers by each disk sending a next sequential block of data to the subscriber, one at a time. As previously stated, the preferred embodiment of the present invention uses declustered mirroring in order to guarantee a data stream at a constant rate to a subscriber. In this context, "mirroring" refers to storing both a primary copy of a block of data and a secondary copy of a block of data where each copy of the block of data is stored on a separate storage device. The term "declustered" refers to dividing the secondary block of data into a number of sub-blocks where each sub-block is stored on a separate storage device. By placing the sub-blocks across many storage devices, when the storage device containing the primary block fails, the burden of transmitting the secondary block of data is shared among many storage devices, thereby lessening the effect of failure mode processing on each storage device. By using declustered mirroring, the preferred embodiment of the present invention guarantees that one component, either a storage device or a server of a storage device, can fail and the data stream is unaffected. A "server" of a storage device is responsible for managing the storage device. As will be described in further detail below, the preferred embodiment can tolerate more than one component failure under certain circumstances.

In addition to utilizing declustered mirroring, the preferred embodiment reserves bandwidth so as to be able to guarantee data streams to subscribers at a constant rate. The term "bandwidth" is intended to refer to the input/output capacity (for a fixed time frame) of storage devices that hold data for video image sequences. The video-on-demand system of the present invention will be described below relative to an implementation that concerns output bandwidth (i.e., reading data from storage devices holding video image sequences), but those skilled in the art will appreciate that the present invention may also be applied to input bandwidth as well (i.e., writing video image sequence data to storage devices). The preferred embodiment reserves bandwidth for both normal mode processing and failure mode processing. Normal mode processing refers to a mode of operation of the video-on-demand system wherein no component failures occur and failure mode processing refers to a mode of operation of the video-on-demand system wherein a component failure occurs. The preferred embodiment allocates a time slot to each subscriber request for video image sequences. This time slot is representative of a bandwidth unit (i.e., a unit of system bandwidth) of the video-on-demand system and is divided into two parts: a primary period and a secondary period. The primary period may be viewed as reserved bandwidth for normal mode processing and the secondary period can be thought of as reserved bandwidth for failure mode processing. The secondary period is typically not used for sending data in normal mode processing of the preferred embodiment. Instead, the secondary period is used in failure mode processing for sending sub-blocks of data in order to compensate for the failure of a component. Thus, by reserving bandwidth for failure mode processing, the data stream to a subscriber is unaffected when a failure occurs.

Further, the preferred embodiment of the present invention reduces the size of overall time slot necessary for normal mode and failure mode processing. That is, the preferred embodiment has a technique ("storage device segmentation") for reducing the amount of time that must be reserved for both normal mode processing and failure mode processing, thereby increasing the total bandwidth of the system. The preferred embodiment reduces the overall time slot by dividing the storage devices into a primary portion and a secondary portion. The primary portion of the storage device contains the primary blocks of data and the secondary portion of the storage device contains the sub-blocks of data. The preferred embodiment designates the primary portion as the faster region (typically the outer region) of the storage device and designates the secondary portion as the slower region (typically the inner region) of the storage device. Thus, the preferred embodiment takes advantage of the increased data transfer rates on the faster regions of a storage device. That is, by using storage device segmentation, the majority of data transferred during a time slot is retrieved from the outer region of the storage device that has a faster data transfer rate than the inner region of the storage device. This technique exploits the fact that storage devices, such as hard disks, typically have a platter with many concentric tracks. The outermost tracks are larger than the inner tracks and thus can store more data. In addition, the platter spins at a constant rate. Thus, in one revolution of the platter, the outermost tracks can transfer more data than the inner tracks. Therefore, the outermost tracks have a faster data transfer rate than the inner tracks. Although storage device segmentation has been described relative to a hard disk, one skilled in the art will appreciate that storage device segmentation can be used with any device having a faster region and a slower region.

In summary, the preferred embodiment of the present invention guarantees data streams to subscribers at a constant rate. In order to do this, the preferred embodiment stores data using declustered mirroring and reserves bandwidth up front for both normal mode processing and failure mode processing. Further, the preferred embodiment uses storage device segmentation to reduce the amount of bandwidth that must be reserved for normal mode processing and failure mode processing. By reducing the amount of bandwidth that must be reserved, the video-on-demand system is more efficient and can service more subscribers. Although the preferred embodiment of the present invention is described below with reference to a video-on-demand system, one skilled in the art will appreciate that the present invention can be used with any continuous media server system or, more generally, any system wherein a data stream must be delivered at a constant rate.

In describing the preferred embodiment of the present invention, the description is presented in three parts. First, a description of the hardware components is presented. Second, a description of the data structures used by the preferred embodiment is presented. Third, the step-by-step processing of the preferred embodiment is presented with accompanying flowcharts to illustrate the interrelationships between the hardware components and the data structures as well as to illustrate overall processing of the preferred embodiment of the present invention.

With respect to the hardware components, the preferred embodiment of the present invention is adapted for use in a video-on-demand server system like that shown in FIG. 1. The system depicted in FIG. 1 is a video-on-demand server system in which subscribers may request at any point in time to view particular video image sequences transmitted from the cable station 10. The cable station 10 transmits the data for the video image sequences over the interconnection network 12 to the subscribers 14. The interconnection network 12 may be any suitable interconnection mechanism, including an asynchronous transfer mode (ATM) network. Functionally, the interconnection network 12 acts like a crosspoint, banyan or other switch topology. The cable station 10 preferably makes available a large number of different video image sequences that may be transmitted to the subscribers 14 and viewed in real time. The data for the video image sequences may contain video data, audio data and other types of data, such as closed captioning data. The present invention may also be applied solely to audio data or other types of data sequences.

For such a video-on-demand server system, the choice of video image sequence viewed by a subscriber is not prescheduled. Viewing choices are scheduled upon subscriber demand. A subscriber need not choose a video image sequence that other subscribers are watching; rather, the subscriber may choose from any of the available video image sequences. Furthermore, each subscriber chooses when he wishes to start viewing a video image sequence. A number of different subscribers 14 may be concurrently viewing different portions of the same video image sequence. A subscriber may select where in a sequence he desires to start viewing and can stop watching a sequence before the entire sequence has been viewed.

FIG. 2 is a block diagram showing the cable station 10 in more detail. The cable station 10 is a video-on-demand server. The cable station 10 includes a controller 16 that is responsible for scheduling transmission of video image sequences to subscribers 14 (FIG. 1). The controller 16 controls several subsystems 18A, 18B, and 18C and is responsible for scheduling and directing output from the subsystems to subscribers 14. The controller may be duplicated to provide a backup controller that enhances the fault tolerance of the system. In addition, one skilled in the art will appreciate that the functioning of the controller can be distributed across the subsystems, thereby eliminating the need for a controller. Although only three subsystems are shown in FIG. 2, those skilled in the art will appreciate that, in most instances, it is more suitable to employ a larger number of subsystems. Only three subsystems are shown in FIG. 2 for purposes of simplicity and clarity.

Each subsystem 18A, 18B, and 18C includes a microprocessor 20A, 20B, and 20C that is responsible for controlling respective pairs of storage devices (22A, 24A), (22B, 24B) and (22C, 24C). The data for the video image sequences that are available to the subscribers 14 are stored on the storage devices 22A, 24A, 22B, 24B, 22C and 24C. Each subsystem 18A, 18B, and 18C need not include two storage devices, rather each subsystem may include only one storage device or may, alternatively, include more than two storage devices. The microprocessors 20A, 20B, and 20C are responsible for cooperating with the controller 16 to transmit the data for the video image sequences stored on the storage devices to the subscribers 14.

Storage devices 22A, 22B, 22C, 24A, 24B and 24C may be, for instance, magnetic disk drives or optical disk drives. Those skilled in the art will appreciate that any suitable storage device may be used for storing the data for the video image sequences. For instance, RAM, masked ROM, EPROM and flash EPROMs may be used to store the video image sequences in the present invention.

FIG. 3A depicts a portion of storage device 22A of FIG. 2 in more detail. Storage device 22A is described with reference to being a disk storage device. Although storage device 22A is depicted, the other storage devices 24A, 22B, 24B, 22C, and 24C are similar. Storage device 22A has an outer region 302 and an inner region 304. The outer region 302 is also known as the primary portion and the inner region 304 is also known as the secondary portion. As previously described, the data transfer rates for the outer region 302 far exceed those of the inner region 304. Also, as previously described, the video image sequences are divided into sequential blocks of data that are striped across the primary portions of the storage devices. Block size is variable, but typically a block includes 64 kilobytes to 4 megabytes of data. Block size is bounded by an upper limit that may not be exceeded. Striping the blocks of data refers to storing a first block of data on a first storage device and each sequentially following block of data is stored on the next sequential storage device. When reaching the last storage device, the preferred embodiment wraps around and stores the next block of data on the first storage device. This striping continues until all the blocks of data are stored across the storage devices. By storing the blocks on the primary portion of a storage device, it guarantees faster data transfer rates for the majority of the data that a storage device transfers.

After storing the primary blocks of data on the primary portions of the storage devices, the preferred embodiment of the present invention then stores data onto the secondary portions of the storage devices by utilizing declustered mirroring. Although the preferred embodiment is described as storing data on the secondary portions of the storage devices after storing data on the primary portions of the storage devices, one skilled in the art will appreciate that data can be stored on the primary portions after the secondary portions or data can be stored on the primary portions and the secondary portions simultaneously. The data on the

secondary portion is used during failure mode processing. For each block of data on the primary portion of a storage device, the block of data is divided into "D" sub-blocks, where "D" refers to a declustering number. That is, the declustering number is the number of storage devices across which the sub-blocks are stored. As the declustering number is increased, the number of storage devices that are used for transmitting the sub-blocks of data during failure mode processing is increased, which lessens the burden of performing failure mode processing by each storage device. However, the greater the declustering number, the greater the ratio of network and other system overhead to the amount of data transferred. Although the preferred embodiment of the present invention uses a declustering number of 8, one skilled in the art will appreciate that other declustering numbers can be used by the present invention. Another benefit associated with using a higher declustering number is that as the declustering number is increased, the amount of bandwidth that is reserved for failure mode processing is reduced (i.e., the secondary period of the time slot). Therefore, since the primary period of the time slot transfers data at a faster rate than the secondary period of the time slot, a greater declustering number means less data is being transferred from the slower part of the storage device and thus a faster overall data transfer rate is realized. In turn, the faster the data transfer rate, the smaller the amount of bandwidth that must be reserved by the system and the more subscribers can be serviced by the system.

FIG. 3B depicts a more derailed diagram of storage device 22A of FIG. 2 in an alternative embodiment of the present invention. Storage device 22A is described with reference to being a disk storage device. In the alternative embodiment, the innermost region of the storage device 22A is an unused region 305. The unused region 305 has the slowest data transfer rate of the storage device 22A and is thus unused so as to increase the data transfer rate of the overall storage device. The outer region 302 and the inner region 307 are accordingly smaller in size.

FIG. 3C depicts an example of declustered mirroring of the preferred embodiment of the present invention. FIG. 3C depicts three storage devices 306, 308, 310 with each storage device having a primary portion 312, 316, 320 and a secondary portion 314, 318, 322. In this example, the video image sequences are comprised of three blocks of data, block A, block B, and block C which are stored on the primary portions 312, 316, 320 of the storage devices, respectively. In this example, the declustering number is 2 and, therefore, block A is divided into two sub-blocks with the first sub-block A1 being stored on the secondary portion 318 of storage device 308 and the second sub-block A2 being stored on the secondary portion 322 of storage device 310. Block B is divided into two sub-blocks, B1 and B2, which are stored on the secondary portions of storage devices 310, 306, respectively. Also, block C is divided into two sub-blocks, C1 and C2, which are stored on the secondary portions of storage devices 306, 308, respectively. Therefore, by striping the data on the primary portions of the storage devices and storing the sub-blocks on the secondary portions of the storage devices, if a failure occurs to storage device 308, storage device 310 and storage device 306 can each send sub-blocks B1 and B2 so that the data stream to the subscriber is not interrupted. Although a video image sequence has been described as comprising three blocks of data, one skilled in the art will appreciate that a video image sequence can comprise many blocks of data. In addition, although only one video image sequence has been described as being striped across the storage devices, one skilled in the art will appreciate that additional video image sequences can be stored in this manner by the present invention.

By utilizing declustered mirroring as shown in FIG. 3C, the video-on-demand system of the present invention can tolerate a storage device failure and continue operating in a seamless manner (i.e., without interruption). However, the preferred embodiment of the present invention can also tolerate the failure of a subsystem in a seamless manner without interrupting the data stream to the subscriber. With respect to transferring data, the preferred embodiment just treats the storage devices of the failed subsystem as having failed.

In order to tolerate the failure of a subsystem, the preferred embodiment of the present invention assigns numbers to the storage devices in a particular manner. The numbers are assigned by first sequentially numbering each subsystem from 1 to N. The number assigned to a subsystem can be expressed by the variable "i." Each storage device for a subsystem i is then assigned a number as follows: i, n+i, 2n+i . . . until all storage devices are numbered. For example, when numbering the storage devices of FIG. 2, storage device 24A may be considered the first storage device, with storage device 24B being the second, storage device 24C being the third, storage device 22A being the fourth, storage device 22B being the fifth, and storage device 22C being the sixth. Therefore, if a declustering number of 2 is used with the system depicted in FIG. 2 and with the storage devices being numbered as previously described, the blocks on storage device 24A are divided into sub-blocks that are stored on storage device 24B and storage device 24C. Further, the blocks stored on storage device 22A are stored as sub-blocks on storage device 22B and storage device 22C. Therefore, if subsystem 18A were to fail, the storage devices of subsystems 18B and 18C are able to transmit the data that would have been transmitted by the storage devices of subsystem 18A and, therefore, the data stream to the subscribers is not interrupted.

Although the preferred embodiment of the present invention has been described as tolerating the failure of one storage device or one subsystem, one skilled in the art will appreciate that as the number of subsystems increases and the number of storage devices increases, a declustering number can be chosen so that more than one storage device or subsystem can fail without interrupting the data stream to the subscribers. That is, in the preferred embodiment of the present invention, if failed storage devices or subsystems are spread out with no less than "D" storage devices between the failed components, no interruption of data streams to subscribers occurs.

The declustered mirroring of the present invention has alternative embodiments of which two are described below. The first alternative embodiment spreads the burden of performing failure mode processing across more storage devices than the preferred embodiment, thereby lessening the effect of failure mode processing on any one storage device. The first alternative embodiment sequentially numbers each subsystem from 1 to N. Then, for each subsystem 1 to N, each storage device is sequentially numbered. For example, if subsystem 1 had three storage devices, these storage devices would be numbered 1, 2 and 3, respectively. The second subsystem would then number its storage devices starting with the number 4 and so on until all of the storage devices for each of the subsystems are numbered. After numbering all of the storage devices in this manner, the blocks for the lowest numbered storage device for a subsystem are stored on the secondary portion of the lowest numbered storage device for the "D" subsystems that follow

the subsystem. The blocks on the primary portion of the next sequentially numbered storage device on the subsystem are split into D sub-blocks and are then stored on the next to lowest numbered storage device of the D+1 through 2D subsystems that follow the subsystem. Therefore, the blocks on the lowest numbered storage device would be stored across the D following subsystems on the lowest numbered storage device of each subsystem and the blocks on the next sequential storage device would be stored across the D+1 through 2D following subsystems on the next to lowest numbered storage device of each subsystem. This process is continued until all blocks on each storage device are stored as subblocks.

The first alternative embodiment is perhaps best described by way of an example, which is provided in FIG. 3D. In FIG. 3D, there are five subsystems, subsystem **1**, subsystem **2**, subsystem **3**, subsystem **4** and subsystem **5** with each subsystem having two storage devices, SD1, SD2, SD3, SD4, SD5, SD6, SD7, SD8, SD9 and SD10. Each storage device has a primary portion for storing blocks and a secondary portion for storing sub-blocks. In this example, a declustering number of two is used. As can be seen in FIG. 3D, block A, stored on the primary portion of SD1, is divided into two sub-blocks, A1 and A2, which are stored on the secondary portions of SD3 and SD5, respectively. Block B, stored on SD2, is divided into two sub-blocks, B1 and B2, which are stored on the secondary portions of storage devices SD8 and SD10, respectively. By utilizing the first alternative embodiment, the burden of performing failure mode processing is divided amongst many disks. For example, when subsystem **1** fails, the load for performing failure mode processing is equally distributed over subsystems **2**, **3**, **4** and **5**, and storage devices SD3, SD5, SD8 and SD10.

The second alternative embodiment of declustered mirroring of the present invention reduces the vulnerability of the system to the failure of two or more components. As previously stated, the preferred embodiment can tolerate a second component failure if the second component is not within D storage devices of the component that failed. That is, the system cannot tolerate a component failure within the following "D" storage devices or the preceding "D" storage devices from the component that failed. The preferred embodiment cannot tolerate the failure of a component within D following storage devices since the blocks for a storage device are stored on the D following storage devices. The preferred embodiment cannot tolerate the failure of a second component within D preceding storage devices since a storage device stores sub-blocks for the D preceding storage devices. Therefore, the preferred embodiment is vulnerable to the failure of 2D storage devices.

The second alternative embodiment of declustered mirroring reduces the vulnerability of the system to the failure of two or more components by dividing the storage devices into groups of clusters. A "cluster" is a group of storage devices containing D+1 storage devices. For each block on a storage device in a cluster, the block is divided into D sub-blocks and is stored on the other storage devices within the cluster. As such, by utilizing the second alternative embodiment, if a storage device fails within a cluster, the system can continue operating without interruption even if a second storage device fails, as long as the second storage device is not within the cluster of the failed storage device. Therefore, the second alternative embodiment is vulnerable to the failure of D+1 storage devices and, as such, increases the tolerance of the system for multiple failures. An example of the second alternative embodiment is depicted in FIG. 3E. In this figure, there are six storage devices, SD1, SD2, SD3, SD4, SD5, SD6 that are divided into two clusters, cluster **1** and cluster **2**. The declustering number used in this example is 2. As can be seen from the figure, the block stored on the primary portion of a storage device within a cluster is divided into sub-blocks that are stored on the secondary portions of the other storage devices within the cluster. For example, block A is stored on the primary portion of SD1 and is divided into two sub-blocks, A1 and A2, which are stored on the secondary portions of SD2 and SD3, respectively. Similarly, block B stored on the primary portion of SD2, is divided into sub-blocks B1 and B2, which are stored on the secondary portions of SD3 and SD1, respectively. Furthermore, block C, stored on the primary portion of SD3, is divided into sub-blocks C1 and C2, which are stored on the secondary portions of SD1 and SD2, respectively.

Although two alternative embodiments have been described, one skilled in the art will appreciate that other numberings or groupings of the storage devices can be used by the present invention. Further, one skilled in the art will appreciate that both the blocks and sub-blocks can be stored in a different manner by the present invention.

With respect to the data structures used by the preferred embodiment, scheduling for each storage device is done on a column of time slots. Each column includes a number of time slots in a sequence that repeats. Each time slot is a bounded period of time that is sufficient for the storage device to output a block of data. One time slot from each column of time slots together comprise a bandwidth unit. A bandwidth unit is a unit of allocation of bandwidth of the video-on-demand system of the present invention and is used to transfer data. Each time slot in the bandwidth unit is associated with a different storage device that outputs a block of data of a video image sequence. Since the blocks of data are striped across the storage device, consecutive blocks of data are read from the predetermined sequence of storage devices during the sequence of time slots of the bandwidth unit. The time slots are generated by the controller **16** or other suitable mechanism (FIG. 2).

The notions of a column of time slots and a bandwidth unit can perhaps best be explained by way of example. Subscribers are scheduled by bandwidth unit. In other words, they are granted the same numbered time slot in each column. FIG. 4 shows the scheduling of seven subscribers for three storage devices (e.g., disk **1**, disk **2** and disk **3**). The rectangles (e.g., **400**) shown in FIG. 4 are time slots. Each time slot has a primary period (e.g., **402**) and a secondary period (e.g., **404**). The primary period of the time slots is for sending data from the primary portion of the storage device and the secondary portion of the time slot is for sending data from the secondary portion of the storage device. The numbers 1–7 in FIG. 4 correspond to the time slot in the respective columns **1**, **2** and **3**. Time slots of a common bandwidth unit all have the same number. Columns **1**, **2** and **3** are all offset temporally relative (i.e., a time unit in FIG. 4) to each other, but each column has the same sequence of time slots. As can be seen in FIG. 4, disk drive **1** services each of the subscribers in sequence beginning with the subscriber who has been allocated logical unit of bandwidth **1**. In the example of FIG. 4, bandwidth unit **1** includes the time slots labeled **1** in columns **1**, **2** and **3**. During the slot **1** of column **1**, disk drive **1** begins outputting a block of data for a video image sequence to a first subscriber that has been assigned bandwidth unit **1**. One time unit later, disk drive **2** outputs the next block of data to the first subscriber during time slot **1** of column **2**. Further, at time unit **2**, disk drive **3** outputs the next block of data for the video image sequence

to the subscriber during time slot 1 of column 3. The predefined sequence of storage devices in this example is disk drive 1, disk drive 2 and disk drive 3, with the sequence wrapping back around to disk drive 1 from disk drive 3.

FIG. 5 depicts the columns of time slots of FIG. 4 after a failure of disk 2 has been detected. After a failure is detected by the preferred embodiment of the present invention, the blocks that would normally be sent by the storage device that failed are sent as sub-blocks by the "D" following disks. For example, FIG. 5 depicts an example of disk 2 failing with a declustering number of 2. That is, all of the blocks contained on the primary portion of disk 2 are stored as sub-blocks on the secondary portions of disk 1 and disk 3. During the primary period of the time slots for both disk 1 and disk 3, processing is performed as normal. That is, for example, disk 1 in the primary period of the first time slot sends a block destined for subscriber 1. However, after detecting a failure, the secondary periods of the time slots for both disk 1 and disk 3 are used for sending the sub-blocks that when combined comprise the block ("aggregate block") that should have been sent by the failed disk. For example, disk 2 during the first time slot was to send a block destined for subscriber 6. Since disk 2 has failed, during the secondary period of the first time slot of disk 1, disk 1 sends the first sub-block of the block destined for subscriber 6 (e.g., 6.0). In addition, during the secondary period of the first time slot of disk 3, disk 3 sends the second sub-block that is to be sent to subscriber 6 (e.g., 6.1). Therefore, using this method, subscriber 6 receives the block of data as an aggregate block without an interruption in the data stream. In other words, the data stream to all the subscribers scheduled for disk 2 will be uninterrupted when a failure occurs of disk 2. To the subscriber, no interruption in service is noticed and therefore the subscriber is unaware that a failure has occurred.

With respect to the step-by-step processing performed by the preferred embodiment, FIG. 6 depicts a flowchart functionally illustrating the steps performed by the preferred embodiment of the present invention. The preferred embodiment of the present invention is responsible for assigning numbers to the storage devices, storing data on the storage devices, receiving subscriber requests, scheduling the subscriber requests, and transmitting blocks of data in sequence to the subscribers during both normal mode processing and failure mode processing. The first step performed by the preferred embodiment of the present invention is to assign a number to each storage device (step 601). In this step, the preferred embodiment assigns a number to each storage device as previously described where a sequential number is assigned to one storage device of each subsystem. After assigning a sequential number to one storage device of each subsystem, the preferred embodiment wraps around and then assigns a sequential number to a second storage device of each subsystem. This process continues until all storage devices are assigned a number. After assigning a number to each storage device, the preferred embodiment stripes the data across the storage devices (step 602). In this step, the preferred embodiment of the present invention stripes all the blocks for one or more video images across the primary portions of the storage devices. In addition, the preferred embodiment divides each block into "D" sub-blocks and stores the sub-blocks for a particular block on the "D" numerically following storage devices. This step will be described in greater detail below. After striping the data across the storage devices, the preferred embodiment receives subscriber requests (step 604).

After receiving subscriber requests, the preferred embodiments schedules the subscriber requests (step 606). In this step, the preferred embodiment determines the storage device on which the initial block to be viewed in the video image sequence is stored for a particular subscriber. If the subscriber is viewing the video image sequence from the beginning of the sequence, the initial block is the first block in the sequence. However, where the subscriber desires to view the video image sequence beginning at some intermediate point, the initial block is the first block that the subscriber desires to view. Once the storage device that holds the initial block of the requested video image sequence to be viewed has been identified, the preferred embodiment of the present invention selects a bandwidth unit that may be used to transmit the video data of the requested video image sequence to the requesting subscriber. The preferred embodiment of the present invention selects the next bandwidth unit that is available (i.e., unallocated to a subscriber). The scheduling of subscriber requests and, more generally, the video-on-demand system of the present invention is more clearly described in U.S. patent application Ser. No. 08/159,188, entitled "Method and System for Scheduling the Transfer of Data Sequences," which is hereby incorporated by reference. Alternatively, the preferred embodiment of the present invention may schedule subscriber requests as described in U.S. patent application Ser. No. 08/349,889, entitled "Method and System for Scheduling the Transfer of Data Sequences Utilizing an Anticlustering Scheduling Algorithm," which is hereby incorporated by reference. After scheduling subscriber requests, the preferred embodiment of the present invention transmits blocks of data in sequence to the subscribers (step 608). In this step, the preferred embodiment accesses the columns of time slots and transmits the blocks of data to the subscribers. In addition, if a component fails, the preferred embodiment switches to failure mode and continues transmitting blocks of data in sequence to the subscribers without the subscribers noticing a disruption in the data stream. With regard to a particular video image sequence, the blocks of data are transmitted until either the end of the video image sequence or until the subscriber requests the video image sequence to stop. This step is described in more detail below. Although the steps of FIG. 6 have been described with a specific order, one skilled in the art will appreciate that two or more of the steps may be performed concurrently or in a different order. For example, while the preferred embodiment is transmitting blocks of data in sequence, the preferred embodiment can receive more subscriber requests and schedule those subscriber requests.

FIG. 7 depicts a flowchart of the steps performed by the preferred embodiment of the present invention when striping data for a video image across the storage devices. Although the striping of data is described for only one video image, one skilled in the art will appreciate that the present invention can stripe many video images across the storage devices. The first step performed by the preferred embodiment when striping data is to select the next storage device, starting with an arbitrary storage device (step 702). In this step, the preferred embodiment selects the next storage device for storing a block of data or upon the first invocation of this step, the preferred embodiment selects an arbitrary storage device. When the preferred embodiment selects the next storage device, if the last storage device is encountered, the preferred embodiment wraps around and selects the first storage device. Alternatively, instead of selecting an arbitrary storage device, one skilled in the art will appreciate that the storage device that is least full may be initially selected by the present invention. After selecting the next storage device, the preferred embodiment selects the next block of

data, starting with the first (step **704**). That is, the preferred embodiment selects the next block of data from the video image to be stored or the first block of data if this step is being invoked for the first time. After selecting the next block of data, the preferred embodiment stores the selected block on the primary portion of the selected storage device (step **706**). After storing the block of data on the primary portion of the storage device, the preferred embodiment divides the block into "D" sub-blocks and stores the sub-blocks on the secondary portion of the next sequential "D" storage devices (step **708**). In this step, after dividing the block into sub-blocks, the sub-block corresponding to the first part of the block is stored on the next sequential storage device and each subsequent sub-block is stored on a sequentially following storage device. After dividing and storing the sub-blocks on the secondary portions of the storage devices, the preferred embodiment determines whether there are more blocks in the video image to be stored (step **710**). If there are more blocks to be stored, the preferred embodiment continues to step **702** wherein the preferred embodiment selects the next sequential storage device. However, if all of the blocks have been stored, processing ends.

FIG. 8 depicts a flowchart of the steps performed by the preferred embodiment of the present invention when transmitting blocks in sequence to subscribers. Steps **804**–**806** reflect the normal mode processing performed by the preferred embodiment of the present invention. Steps **808**–**818** describe the failure mode processing performed by the preferred embodiment of the present invention. The first step performed by the preferred embodiment is to determine if a component has failed (step **802**). In this step, the system detects whether a subsystem has failed or a storage device has failed. The system detects the failure of a subsystem by using a "deadman protocol." In utilizing the deadman protocol, each subsystem sends a ping (i.e., a message) after a predetermined amount of time to the sequentially preceding subsystem and listens to the subsystem that sequentially follows the subsystem. If a subsystem has not received a ping within a predetermined period of time from the sequentially following subsystem, a time-out occurs. Upon the time-out occurring, the subsystem signals the controller and the controller sends a ping to the sequentially following subsystem. If the sequentially following subsystem does not respond to the ping from the controller, the controller determines that the sequentially following subsystem has failed. The detection of a storage device failure occurs when a subsystem detects that one of its storage devices is no longer sending data. After detecting that the storage device is no longer sending data, the subsystem sends a message to the controller indicating the failure of the storage device. If the preferred embodiment does not detect a component failure, processing continues to step **804** and the preferred embodiment performs normal processing.

In performing normal processing, the preferred embodiment receives data from the subsystems (step **804**). In this step, the preferred embodiment accesses the column of time slots and processes subscriber requests for the primary period of each time slot. In effect, each storage device marches down its column of time slots and processes each subscriber request. In processing subscriber requests, the storage devices send the appropriate block of data for a particular subscriber. After receiving the data from the storage devices, the preferred embodiment sends the data to the subscribers (step **806**). In this step, the system determines the subscriber for each block of data received and sends the blocks to the appropriate subscriber via the interconnection network.

If the preferred embodiment detects a component failure, the preferred embodiment realigns the deadman protocol if the component failure detected is a subsystem failure (step **808**). When the preferred embodiment realigns the deadman protocol, it indicates to the immediately following subsystem to send the ping to the immediately preceding subsystem of the failed subsystem. In addition, the immediately preceding subsystem listens for the ping of the immediately following subsystem. After realigning the deadman protocol, the preferred embodiment adjusts the column of time slots for each "D" following storage device (step **810**). This is done by inserting entries into the secondary period of each time slot for each "D" following storage device. The entry in each secondary period corresponds to the entry in the primary period of the same time unit for the storage device that failed. For example, if in time slot one, the failed storage device were to send a particular block to subscriber **6**, the "D" following storage devices send the corresponding sub-block of subscriber **6** during the secondary period of the time slot that they are currently processing when the failed storage device would have been processing the first time slot. In this step, when a subsystem fails, the storage devices for the failed subsystem are treated as having failed.

After adjusting the time slots, the preferred embodiment receives data from the subsystems (step **812**). In this step, the preferred embodiment receives both blocks of data from the primary portion of the storage devices as well as sub-blocks from the secondary portion of the storage devices. After receiving data from the subsystems, the preferred embodiment sends the data to the subscribers (step **816**). The processing of this step is similar to that as described relative to step **806** above, except that upon receiving sub-blocks, the subscribers combine the sub-blocks into aggregate blocks. After sending data to the subscribers, the system determines if there is an additional component failure (step **818**). The processing of this step is similar to that as described relative to step **802** above. If an additional component failure is detected, processing continues to step **808**. However, if an additional component failure is not detected, processing continues to step **812** and the preferred embodiment continues to operate in failure mode. It should be noted that the preferred embodiment operates in failure mode until a system administrator can replace the component that has failed. However, until that time, the video-on-demand system of the present invention continues to deliver data streams to subscribers without the subscribers noticing any interruption in the data streams. Therefore, the preferred embodiment of the present invention sends data to subscribers at a constant rate and can thus guarantee the constant rate in the face of at least one component failure.

While the present invention has been described with reference to a preferred embodiment thereof, those skilled in the art will appreciate that various changes in form and detail may be made without departing from the spirit and scope of the present invention as defined in the appended claims. For instance, other storage media may be used and different quantities of storage media may be used. In addition, different declustering numbers may be used and the ordering of the storage devices may differ.

We claim:

1. A continuous media server system having a consumer for consuming data at a given amount per time interval, the continuous media server system for delivering data to the consumer at the given amount per the time interval, comprising:

a plurality of storage devices containing data, wherein the data comprises blocks and sub-blocks, wherein a block

is divided into a clustering number of sub-blocks, wherein the clustering number is a number greater than one, and wherein sub-blocks for a block on a first storage device are stored on the clustering number of storage devices that follow the first storage device; and

a send component for sending a sequence of the data to the consumer at the given amount per the time interval, wherein the sequence comprises the blocks and when a failure occurs such that a block cannot be sent to the consumer, the sequence comprises the sub-blocks for the block from the clustering number of storage devices that follow the storage device that stores the block to ensure that the sequence of data to the consumer is uninterrupted.

2. The continuous media server system of claim 1 wherein the storage devices are sequential and the sub-blocks are striped across the storage devices.

3. The continuous media server system of claim 1 wherein the storage devices comprise a faster region and a slower region such that the blocks of data are stored on the faster region of the storage device and the sub-blocks of data are stored on the slower region of the storage devices.

4. The continuous media server system of claim 1 wherein the storage devices comprise a fast region, a medium speed region and an unused region such that the blocks of data are stored on the fast region of the storage device, the sub-blocks of data are stored on the medium speed region of the storage device and the unused region comprises a portion of the storage device that has a slower data transfer rate than the fast region and the medium speed region.

5. The continuous media server system of claim 1, further including a reserver component for reserving bandwidth for sending both sub-blocks and blocks to the consumers, wherein bandwidth is output capacity of the system, and wherein the send component sends the blocks from the storage devices to the consumers utilizing the reserved bandwidth and when a storage device fails, the send component sends the sub-blocks utilizing the reserved bandwidth.

6. The continuous media server system of claim 1, further comprising a plurality of subsystems for managing the storage devices and including a means for sending the sub-blocks from the clustering number of storage devices that follow a second storage device when a subsystem that manages the second storage device fails.

7. The continuous media server system of claim 1, further comprising an ordering of subsystems for managing the storage devices and a numbering component for performing a sequence of assigning sequential numbers to the storage devices wherein one storage device is assigned a sequential number from each ordered subsystem and for repeating the sequence until all storage devices are assigned a sequential number.

8. The continuous media server system of claim 1 wherein the system is a video-on-demand system and wherein the data is video image sequences.

9. The continuous media server system of claim 1 wherein the sequence of the data is a stream of the data and wherein the send component sends the stream of the data to the consumer at a constant rate over a period of time.

10. In a video-demand system having a consumer for consuming data at a given amount per time interval, the video-on-demand system for delivering data to the consumer at the given amount per the time interval, the video-on-demand system having a plurality of sequential storage devices for storing data, wherein the data comprises video image sequences having sequential blocks, a method comprising the steps of:

under the control of the video-on-demand system,

storing the blocks on the storage devices such that after storing a block a next sequential block is stored on a next sequential storage device;

dividing the blocks into a clustering number of sub-blocks, wherein the clustering number is a number greater than one; and

for each block,

storing sub-blocks for the block on the clustering number of storage devices that sequentially follow a storage device on which the block is stored.

11. The method of claim 10 wherein the storage devices comprise a faster region and a slower region, wherein the step of storing the blocks includes the step of storing the blocks on the faster region of the storage devices such that after storing a block, a next sequential block is stored on a next sequential storage device, and wherein the step of storing sub-blocks includes the step of storing sub-blocks for the block on the slower region of the clustering number of storage devices that sequentially follow the storage device on which the block is stored.

12. The method of claim 11, further including the step of providing an unused region to the storage devices that is a portion of the storage device having a slowest data transfer rate.

13. The method of claim 10 wherein the storage devices are managed by sequential subsystems and wherein the method further includes the steps of:

performing a sequence of assigning sequential numbers to the storage devices wherein one storage device is assigned a sequential number from each sequential subsystem; and

repeating the sequence until all storage devices are assigned a sequential number.

14. In an on-demand media server system having a consumer for consuming data at a given amount per time interval, a plurality of components and a controller, the components comprising a sequence of storage devices for storing blocks of data and sub-blocks of data and a plurality of subsystems for managing the storage devices, the controller for managing the subsystems, wherein the storage devices comprise a primary portion for storing the blocks and a secondary portion for storing the sub-blocks, wherein the blocks are sequential and each block is divided into a clustering number of sub-blocks, wherein the clustering number is a number greater than one, a method for guaranteeing data delivery to the consumer at the given amount per the time interval, comprising the steps of:

under the control of the controller of the on-demand media server system,

receiving blocks from the primary portion of the storage devices;

sending the received blocks to the consumers;

determining when a component has failed; and

when it is determined that a component has failed,

receiving sub-blocks from the secondary portion of the clustering number of storage devices that sequentially follow the component that failed;

combining the received sub-blocks to create an aggregate block; and

sending the aggregate block to the consumers.

15. The method of claim 14 wherein the storage devices comprise a faster region and a slower region, wherein the primary portion of the storage devices is located on the faster region and the secondary portion of the storage devices is located on the slower region, wherein the step of receiving blocks further includes the step of receiving blocks from the

primary portion on the faster region of the storage devices, and wherein the step of receiving sub-blocks further includes the step of receiving sub-blocks from the secondary portion on the slower region of the clustering number of storage devices that sequentially follow the component that failed.

16. In a continuous media server system having a consumer for consuming data at a given amount per time interval, and a plurality of sequential storage devices for storing sequential blocks of data and sub-blocks of data, a method for guaranteeing data delivery to the consumer at the given amount per the time interval, comprising the steps of:

under the control of the continuous media server system, striping the blocks sequentially across the storage devices;

for each block,

dividing the block into a clustering number of sub-blocks, wherein the clustering number is a number greater than one;

storing the sub-blocks on the clustering number of storage devices that sequentially follow a storage device containing the block;

providing the storage devices with a time slot for sending data, wherein the time slot has a primary period and a secondary period;

during the primary period of the time slot,

sending blocks from storage devices to consumers; determining whether a storage device has failed; and when it is determined that a storage device has failed,

sending sub-blocks from the clustering number of storage devices that sequentially follow the storage device that failed during the secondary period of the time slot.

17. The method of claim 16 wherein the storage devices comprise a faster region and a slower region, wherein the blocks are stored on the faster region and the sub-blocks are stored on the slower region, wherein the step of sending blocks includes the step of sending blocks from the faster region of the storage devices to consumers and wherein the step of sending sub-blocks includes the step of sending sub-blocks from the slower region of the clustering number of storage devices that sequentially follow the storage device that failed.

18. The method of claim 16 wherein the storage devices are managed by sequential subsystems and wherein the method further includes the steps of:

performing a sequence of assigning sequential numbers to the storage devices wherein one storage device is assigned a sequential number from each sequential subsystem; and

repeating the sequence until all storage devices are assigned a sequential number.

19. In a data processing system having a consumer for consuming data at a given amount per time interval, a method for guaranteeing data delivery to the consumer at the given amount per the time interval, comprising the steps of:

providing a continuous media server system to the data processing system for guaranteeing data delivery to the consumer at the given amount per the time interval, the continuous media server system comprising a plurality of sequential storage devices for storing data and a plurality of sequential servers for managing the storage devices, wherein the data comprises sequential blocks;

storing the blocks on the storage devices by the continuous media server system such that after storing a block a next sequential block is stored on a next sequential storage device;

dividing the blocks into a clustering number of sub-blocks by the continuous media server system, wherein the clustering number is a number greater than one; and

storing sub-blocks for a block on a storage device that is managed by a server on a storage device of the clustering number of servers that follow the server by the continuous media server system.

20. The method of claim 19, wherein the step of storing sub-blocks includes the step of storing sub-blocks for a second block on a second storage device that is managed by the server on a storage device of the second clustering number of servers that follow the first clustering number of servers.

21. In a continuous media server system having a consumer for consuming data at a given amount per time interval, and a plurality of sequential storage devices for storing data that are grouped into clusters of storage devices, wherein the data comprises sequential blocks, a method for guaranteeing data delivery to the consumer at the given amount per the time interval, comprising the steps of:

under the control of the continuous media server system,

storing the blocks on the storage devices such that after storing a block a next sequential block is stored on a next sequential storage device;

dividing the blocks into a clustering number of sub-blocks, wherein the clustering number is a number greater than one; and

storing sub-blocks for a block on a storage device within a cluster on a clustering number of storage devices within the cluster.

22. In a video-on-demand system having a consumer for consuming data at a constant rate over a period of time, and a plurality of sequential storage devices for storing data, the data comprising video image sequences having sequential blocks, a method for guaranteeing a stream of data to the consumer at the constant rate, comprising the steps of:

under the control of the video-on-demand system,

storing the blocks on the storage devices such that after storing a block a next sequential block is stored on a next sequential storage device;

dividing the blocks into a clustering number of sub-blocks, wherein the clustering number is a number greater than one;

storing the sub-blocks for each block on the clustering number of storage devices that follow the storage device on which the block is stored;

receiving a request for a stream of the data from the consumer;

determining whether a storage device has failed;

when it is determined that a storage device has failed,

for each block,

if the block is not located on the storage device that failed,

sending the block to the consumer; and if the block is located on the storage device that failed,

sending the sub-blocks for the block to the consumer from the clustering number of storage devices that follow the storage device that failed to ensure that the stream of data is uninterrupted due to the storage device failure; and

when it is determined that a storage device has not failed,

for each block,

sending the block to the consumer.

23. A computer-readable media whose contents cause a continuous media server system to become fault tolerant, the continuous media server system having a consumer for consuming data at a constant rate over a period of time and a plurality of sequential storage devices for storing data, the data comprising blocks, the continuous media media server system for sending data to the consumer at the constant rate over the period of time, by performing the steps of:

storing the blocks on the storage devices of the continuous media server system such that after storing a block, a next sequential block is stored on a next sequential storage device;

dividing the blocks into a clustering number of sub-blocks. the clustering number is a number greater than one;

storing the sub-blocks for each block on the storage devices of the continuous media server system that sequentially follow the storage device on which the block is stored;

receiving a request for the data; and

for each block,

determining if the block is located on a storage device that failed;

if it is determined that the block is located on a storage device that has not failed,

sending the block to the consumer; and

if it is determined that the block is located on a storage device that has failed,

sending the sub-blocks for the block to the consumer to ensure that the constant rate at which the data is sent to the consumer does not change due to the failure of the storage device.

* * * * *

D

THIS PAGE BLANK (USPTO)

APPENDIX D

U.S. Patent No. 4,914,570 to Peacock

United States Patent [19]

Peacock

[11] **Patent Number: 4,914,570**

[45] **Date of Patent: Apr. 3, 1990**

[54] **PROCESS DISTRIBUTION AND SHARING SYSTEM FOR MULTIPLE PROCESSOR COMPUTER SYSTEM**

[75] Inventor: **J. Kent Peacock**, San Jose, Calif.

[73] Assignee: **Counterpoint Computers, Inc.**, San Jose, Calif.

[21] Appl. No.: **223,729**

[22] Filed: **Jul. 21, 1988**

Related U.S. Application Data

[63] Continuation of Ser. No. 907,568, Sep. 15, 1986, abandoned.

[51] Int. Cl.^{4} **G06F 13/00**; G06F 9/30

[52] U.S. Cl. **364/200**; 364/229.2; 364/230.3; 364/280.9

[58] **Field of Search** ... 314/200 MS File, 900 MS File

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,245,306	1/1981	Besemer et al.	364/200
4,354,227	10/1982	Hays Jr. et al.	364/200
4,459,664	7/1984	Pottier et al.	364/200
4,567,562	1/1986	Fassbender	364/200
4,621,318	11/1986	Maeda	364/200

OTHER PUBLICATIONS

K. Thompson, Unix Time-Sharing System: Unix Implementation, The Bell Sys. Tech. Journal, vol. 57, No. 6, pp. 1931-1946, Aug. 1978.

Gary Fielland and Dave Rodgers, 32-bit Computer System Shares Load Equally Among up to 12 Processors, Elect. Design, pp. 153-168, Sep. 6, 1984.

Bach and Buroff, A Multiprocessor Unix System, Usenix, pp. 174-177, Summer 1984.

Goble and Marsh, A Dual Processor VAX 11/780, Sigarch, vol. 10, No. 3, pp. 291-296, Apr. 1982.

Multiprocessor Makes Parallelism Work, Electronics, pp. 46-48, Sep. 2, 1985.

Bach and Buroff, Multiprocessor Unix Operating System, The Bell System Tech. Journal, vol. 63, No. 8, pp. 1733-1749, Oct. 1984.

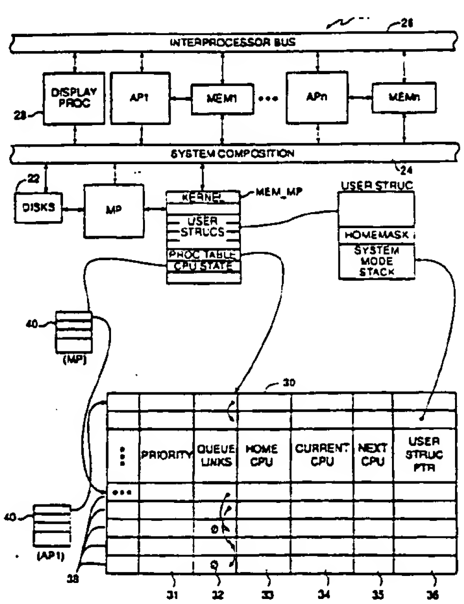
Primary Examiner—Eddie P. Chan

Attorney, Agent, or Firm—Flehr, Hohbach, Test, Albritton & Herbert

[57] **ABSTRACT**

A multiple processor (CPU) computer system, each CPU having a separate, local, random access memory means to which it has direct access. An interprocessor bus couples the CPUs to memories of all the CPUs, so that each CPU can access both its own local memory means and the local memories of the other CPUs. A run queue data structure holds a separate run queue for each of the CPUs. Whenever a new process is created, one of the CPUs is assigned as its home site and the new process is installed in the local memory for the home site. When a specified process needs to be transferred from its home site to another CPU, typically for performing a task which cannot be performed on the home site, the system executes a cross processor call, which performs the steps of: (a) placing the specified process on the run queue of the other CPU; (b) continuing the execution of the specified process on the other CPU, using the local memory for the specified process's home site as the resident memory for the process and using the interprocessor bus to couple the other CPU to the home site's local memory, until a predefined set of tasks has been completed; and then (c) placing the specified process on the run queue of the specified process's home site, so that execution of the process will resume on the process's home site.

13 Claims, 6 Drawing Sheets



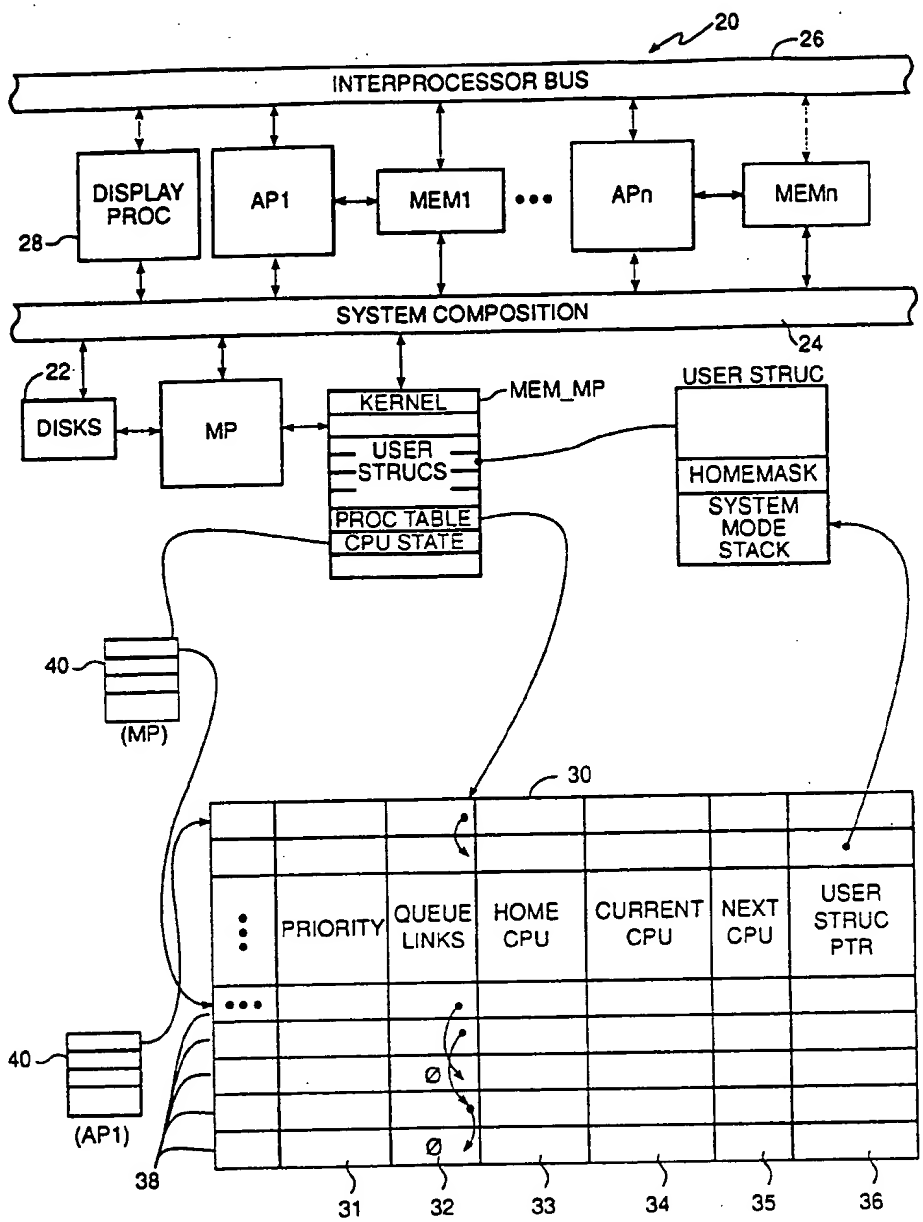

20
26
INTERPROCESSOR BUS
DISPLAY PROC
AP1
MEM1
APn
MEMn
28
SYSTEM COMPOSITION
24
22
DISKS
MP
KERNEL
MEM_MP
USER STRUCS
PROC TABLE
CPU STATE
USER STRUC
HOMEMASK
SYSTEM MODE STACK
40
(MP)
30
PRIORITY
QUEUE LINKS
HOME CPU
CURRENT CPU
NEXT CPU
USER STRUC PTR
Ø
Ø
40
(AP1)
38
31
32
33
34
35
36

FIGURE 1

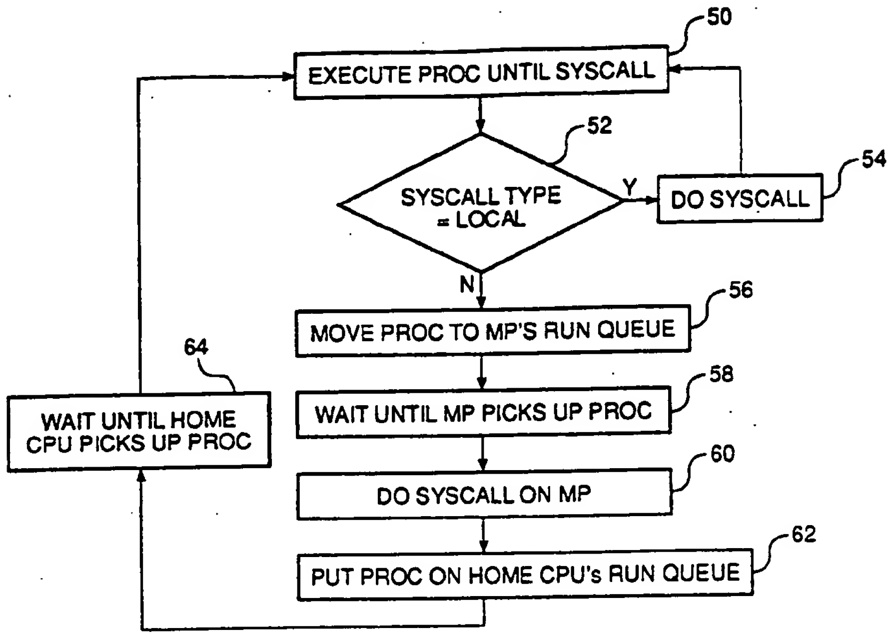


FIGURE 2A

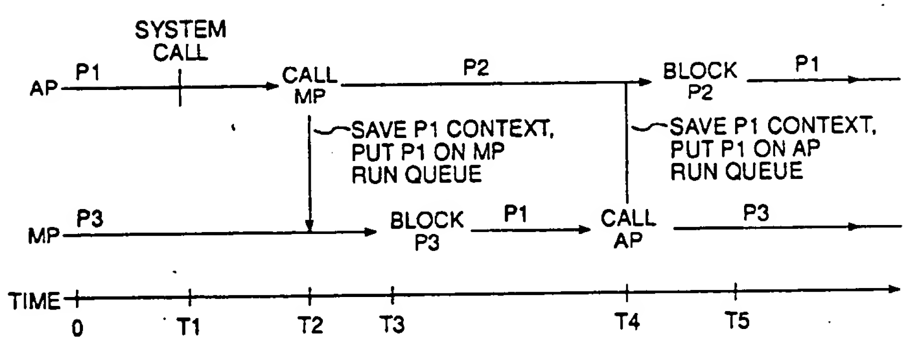


FIGURE 2B

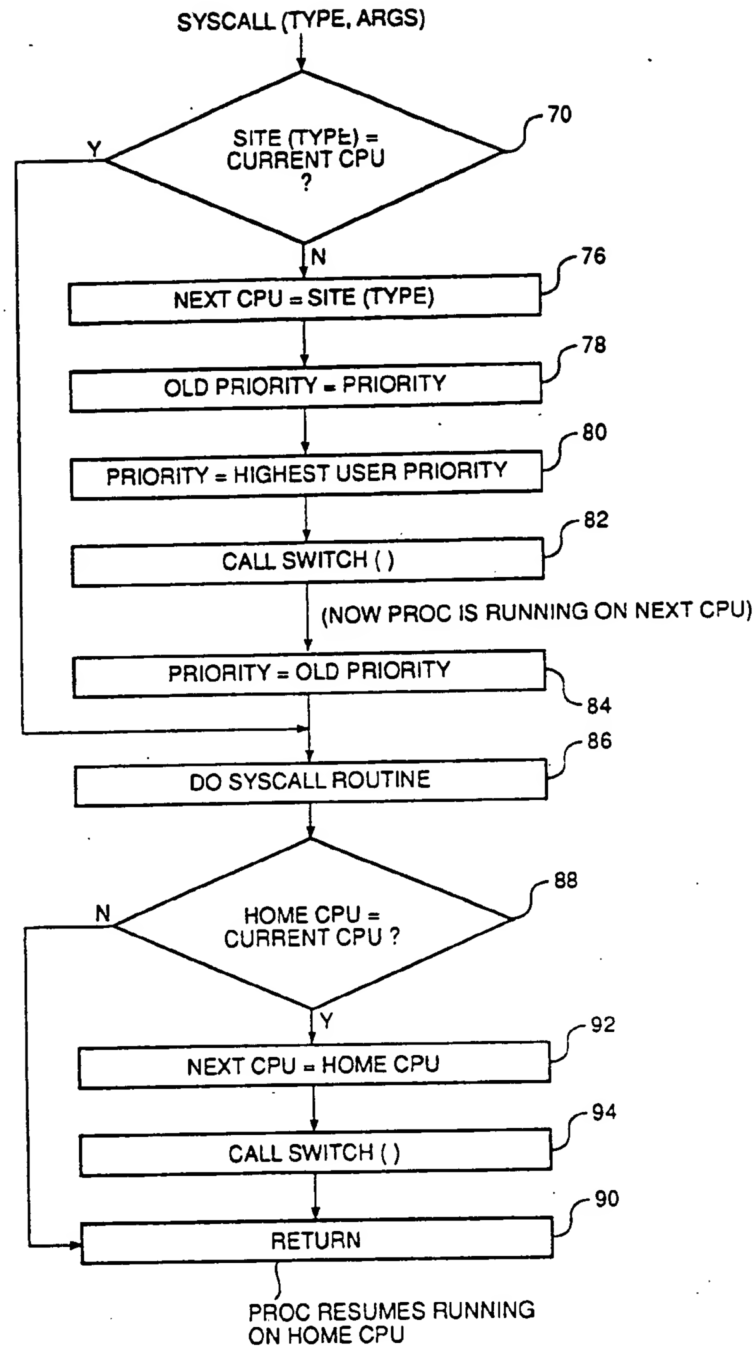

SYSCALL (TYPE, ARGS)
SITE (TYPE) = CURRENT CPU ?
70
Y
N
NEXT CPU = SITE (TYPE)
76
OLD PRIORITY = PRIORITY
78
PRIORITY = HIGHEST USER PRIORITY
80
CALL SWITCH ( )
82
(NOW PROC IS RUNNING ON NEXT CPU)
PRIORITY = OLD PRIORITY
84
DO SYSCALL ROUTINE
86
HOME CPU = CURRENT CPU ?
88
N
Y
NEXT CPU = HOME CPU
92
CALL SWITCH ( )
94
RETURN
90
PROC RESUMES RUNNING ON HOME CPU

FIGURE 4

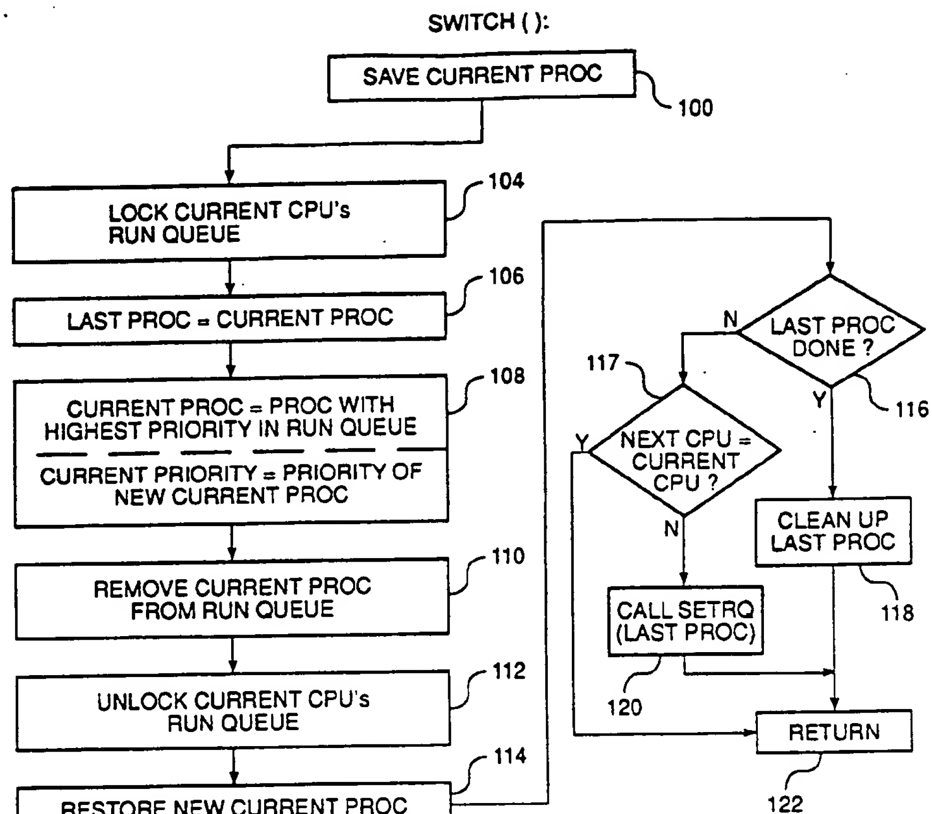

SWITCH ( ):
SAVE CURRENT PROC
100
LOCK CURRENT CPU's RUN QUEUE
104
LAST PROC = CURRENT PROC
106
CURRENT PROC = PROC WITH HIGHEST PRIORITY IN RUN QUEUE
CURRENT PRIORITY = PRIORITY OF NEW CURRENT PROC
108
REMOVE CURRENT PROC FROM RUN QUEUE
110
UNLOCK CURRENT CPU's RUN QUEUE
112
RESTORE NEW CURRENT PROC
114
LAST PROC DONE ?
N
Y
116
117
NEXT CPU = CURRENT CPU ?
Y
N
CLEAN UP LAST PROC
118
CALL SETRQ (LAST PROC)
120
RETURN
122

FIGURE 5

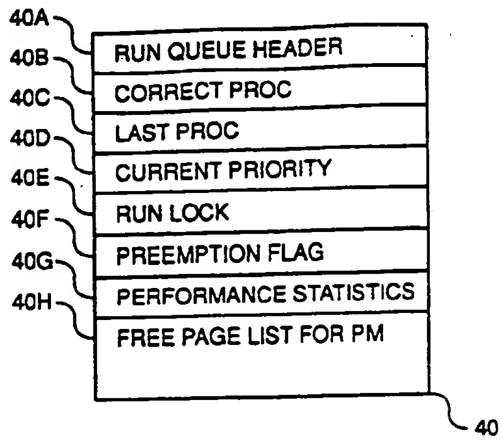


FIGURE 3

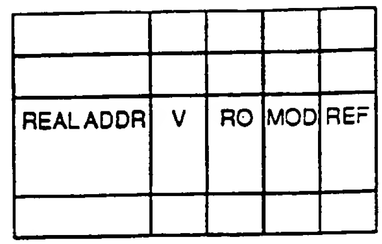


FIGURE 8

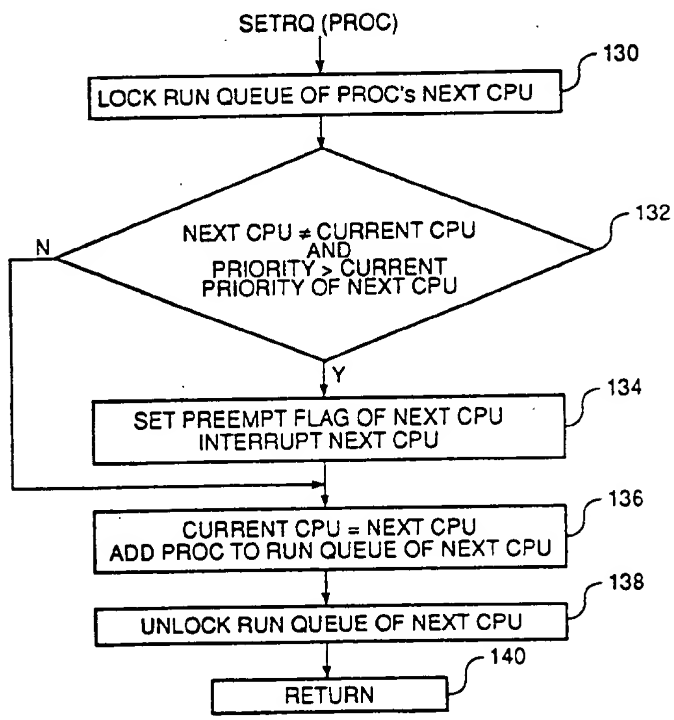


FIGURE 6

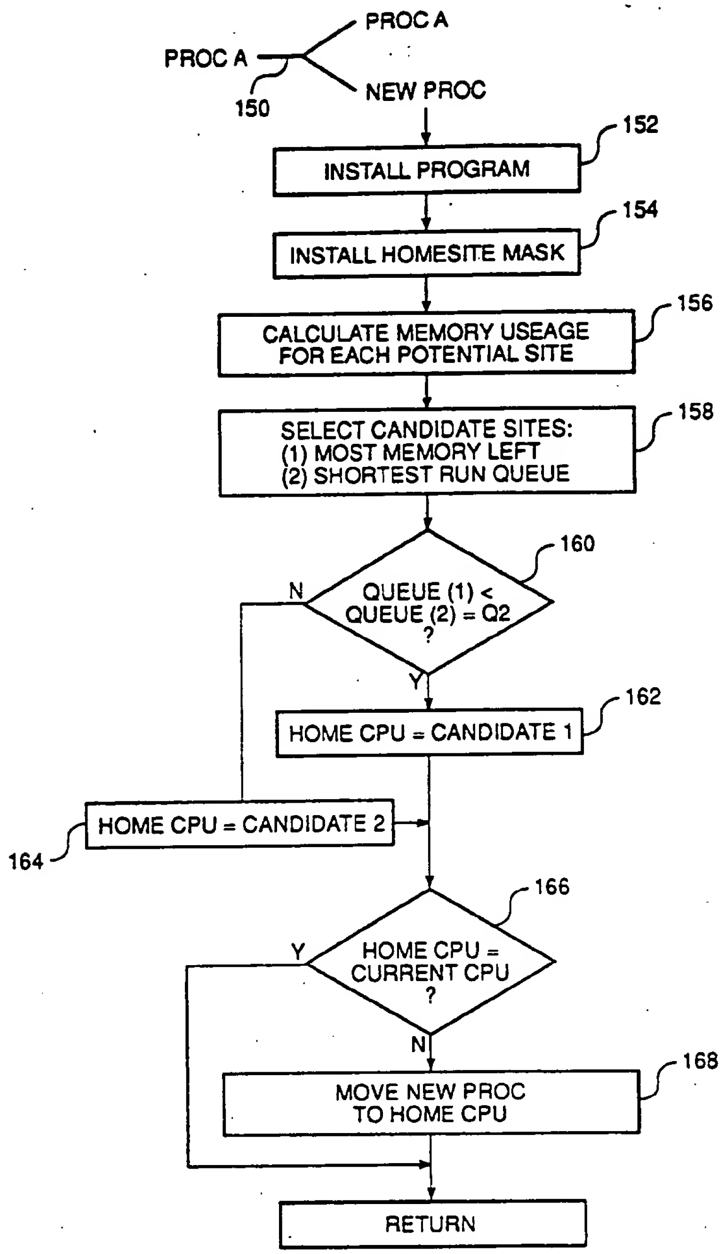

PROC A
PROC A
NEW PROC
150
INSTALL PROGRAM
152
INSTALL HOMESITE MASK
154
CALCULATE MEMORY USEAGE FOR EACH POTENTIAL SITE
156
SELECT CANDIDATE SITES: (1) MOST MEMORY LEFT (2) SHORTEST RUN QUEUE
158
QUEUE (1) < QUEUE (2) = Q2 ?
160
N
Y
HOME CPU = CANDIDATE 1
162
HOME CPU = CANDIDATE 2
164
HOME CPU = CURRENT CPU ?
166
Y
N
MOVE NEW PROC TO HOME CPU
168
RETURN

FIGURE 7

PROCESS DISTRIBUTION AND SHARING SYSTEM FOR MULTIPLE PROCESSOR COMPUTER SYSTEM

This is a continuation of application Ser. No. 907,568 filed Sept. 15, 1986, now abandoned.

The present invention relates generally to multiple processor computer systems, and particularly to apparatus and methods for moving a process from one site to another within a multiple processor computer system.

BACKGROUND OF THE INVENTION

The prior art includes a large number of different multiple processor computer systems, and a number of variations on the UNIX (a trademark of AT&T) operating system.

For the purposes of this introduction, multiple processor computer systems can be generally classified into two distinct types: (1) those that perform complex calculations by allocating portions of the calculation to different processors; and (2) those that are enhanced multitasking systems in which numerous processes are performed simultaneously, or virtually simultaneously, with each process being assigned to and performed on an assigned processor. The present invention concerns the second type of multiple processor system.

In order to avoid confusion between the terms "processor" (which is a piece of apparatus including a central processing unit) and "process" (which is a task being performed by a computer), the terms "site" and "CPU" shall be used herein synonymously with "processor". For instance, when a process is created, it is assigned to a particular site (i.e., processor) for execution.

As background, it should be understood that in any multitasking system, there is a "run queue" which is a list of all the processes which are waiting to run. In most systems the run queue is a linked list. When the system is done with one process (at least temporarily) and ready to start running another process, the system looks through the run queue and selects the process with the highest priority. This process is removed from the run queue and is run by the system until some event causes the system to stop the selected process and to start running another process.

In prior art multiple processor (also called multiple CPU) systems, there is generally a single large memory and a single run queue for all of the processors in the system. While the use of a single run queue is not inherently bad, the use of a single large memory tends to cause increasing memory bus contention as the number of CPUs in the system is increased.

Another problem associated with most multiple CPU computer systems, only one of the CPUs can perform certain tasks and functions, such as disk access. Therefore if a process needs to perform a particular function, but is running at a site which cannot perform that function, the computer system needs to provide a method for that process to perform the function at an appropriate site within the system.

Generally, the problems associated with such "cross processor calls" include (1) minimizing the amount of information which is moved or copied from one site to another each time a process makes a cross processor call; (2) devising a method of updating the system's run queue(s) which prevents two processors from simultaneously changing the same run queue, because this

could produce unreliable results; and (3) providing a method for efficiently transferring a process to a another site and, usually, then automatically transferring the process back to its original site.

The present invention solves the primary memory contention and cross processor call problems associated with prior art multiple CPU systems by providing a separate local memory and a separate run queue for each processor. Memory contention is minimized because most processes are run using local memory. When a process needs to be transferred to a specified processor a cross processor routine simply puts the process on the run queue of the specified CPU. The resident memory for the process remains in the local memory for the process's home CPU, and the specified CPU continues execution of the process using the memory in the home CPU. The process is transferred back to its home CPU as soon as the tasks it needed to perform on the specified CPU are completed.

It is therefore a primary object of the present invention to provide an improved multiple CPU computer system.

Another object of the present invention is to provide an efficient system for transferring processes from one CPU to another in a multiple CPU computer system.

SUMMARY OF THE INVENTION

In summary, the present invention is a multiple processor (CPU) computer system, each CPU having a separate, local, random access memory means to which it has direct access. An interprocessor bus couples the CPUs to memories of all the CPUs, so that each CPU can access both its own local memory means and the memory means of the other CPUs. A run queue data structure holds a separate run queue for each of the CPUs.

Whenever a new process is created, one of the CPUs is assigned as its home site and the new process is installed in the local memory means for the home site. When a specified process needs to be transferred from its home site to another CPU, typically for performing a task which cannot be performed on the home site, the system executes a cross processor call, which performs the steps of: (a) placing the specified process on the run queue of the other CPU; (b) continuing the execution of the specified process on the other CPU, using the memory means for the specified process's home site as the resident memory for the process and using the interprocessor bus means to couple the other CPU to the home site memory means, until a predefined set of tasks has been completed; and then (c) placing the specified process on the run queue of the specified process's home site, so that execution of the process will resume on the process's home site.

BRIEF DESCRIPTION OF THE DRAWINGS

Additional objects and features of the invention will be more readily apparent from the following detailed description and appended claims when taken in conjunction with the drawings, in which:

FIG. 1 is a block diagram of a multiprocessor computer system, and some of its most important data structures, in accordance with the present invention.

FIGS. 2A and 2B schematically represent the cross processor call procedure of the preferred embodiment.

FIG. 3 is a block diagram of the CPUSTATE data structure used in the preferred embodiment of the present invention.

FIG. 4 is a flow chart of the process by which a system subroutine call may cause a process to be moved from one site to another in a computer system.

FIG. 5 is a flow chart of the context switching method used in the preferred embodiment of the invention to move a process from one site to another in a computer system.

FIG. 6 is a flow chart of the subroutine SETRQ which is used by the context switching method diagramed in FIG. 4.

FIG. 7 is a flow chart of the method used for creating a new process and assigning it a home site.

FIG. 8 is a block diagram of a virtual memory management page table used in the preferred embodiment of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring to FIG. 1, there is shown a block diagram of a multiprocessor computer system 20, and some of its most important data structures, in accordance with the present invention. In a typical configuration, the system 20 includes a main processor MP, and a plurality n of application processors APl to APn.

All of the system's processors are homogeneous, separate one-board microcomputers using a Motorola 68020 central processing unit. For convenience, these processors are also called "CPUs" and "sites".

The system 20 is a multitasking computer system which typically has a multiplicity of processes, also called "user processes" or "tasks", running and waiting to run. As will be described in greater detail below, each user process is assigned a "home site" or "home processor" when it is created.

Cross Processor Calls

The present invention provides a simple system for temporarily moving a user process away from its home site to a specified remote site. To do this, the state of the process is saved (just as it would be saved whenever the process is suspended in any other multitasking system), and then a pointer to the user process is simply added to the remote site's "run queue" and removed from the list of processes running on the home site.

When the remote site picks up this user process from its run queue, it merely reinstates the process and runs the process just like any other process. After the task which required the cross processor call is completed, the interprocessor transfer is reversed by saving the user process's state and adding a pointer to the user process to the run queue for its home site.

The system's memory is organized so that the remote site can use the user process's memory at its home site, rather than moving the process's resident memory to the remote site.

There are several advantages to this approach. The first is that the context which requires a user process to be moved away from its home site need not be copied into a message encapsulating the request, and all references to parameters needed by the process can be made directly to the process's resident memory at the home site. Secondly, there is no need to synchronize one processor with any other. Third, the cross processor call is virtually transparent to the system and requires very little overhead and minimal modification of the system's operating system. In most instances, the only modification to the operating system required is the insertion of a cross processor call in system routines which must be executed on a specified CPU.

In the preferred embodiment cross processor calls are performed only to have a system request serviced which cannot be serviced at the home site of the process. FIGS. 2A and 2B schematically represent the cross processor call procedure of the preferred embodiment.

Referring to FIG. 2A, the process identified as Proc is assumed to have a home site on an applications processor (i.e., not the main processor) and to be running on that home site. The process runs until a system routine is called (box 50). If the system routine can be run locally (box 52), the system routine is simply executed (box 54) and the process continues to run on the home site (box 50).

If, however, the system routine cannot be run locally (box 52), then the process is put on the main processor's run queue (box 56), where it waits until the main processor MP picks it up for execution (box 58). Then the process resumes running on the MP, where it runs the system routine.

After the system routine completes the tasks which need to be performed by the MP, the process is put back on the home site's run queue (box 62), where it waits until the home CPU is ready to pick it up (box 64) and to continue execution of the process (box 50).

Looking at this same process from another perspective, in FIG. 2B at time zero process P1 is running on processor AP and process P3 is running on the main processor MP. At time t1 process P1 performs a system call (i.e., it calls a system routine) which requires processing on the MP. Therefore, shortly after the system call, at time t2, P1's context is saved (in its user structure) and P1 is put on MP's run queue. It should be noted that P1 is usually given a high priority when it performs a cross processor call so that its request will be serviced quickly. Also, when the AP stops running P1 it picks up another process P2 from its run queue and runs that process.

MP continues to run process P3 until some event blocks or interrupts P3's execution at time t3, at which point the MP will run Pl. At time t4, when P1 finishes the system tasks which required the use of the main processor MP, P1's context is saved and it is put back on the run queue for its home site, AP. At this point the MP picks up the highest priority process from its run queue, which may be the process P3 that was interrupted by P1.

AP continues to run process P2 until some event blocks or interrupts P2's execution at time t5, at which point the AP will pick up the highest priority process from its run queue, which may be the process P1.

Memory

Referring again to FIG. 1, each CPU has its own memory module MEM_MP, and MEMl to MEMn, which is used as the primary random access memory for its corresponding CPU.

The resident memory for each user process in the system is located in the memory module for its home site.

While each CPU has its own memory module, these memory modules are multiported and connected to at least one bus so that all physical memory in the system can be accessed by any processor in the system. That is, all of the system's physical memory resides in a single, large physical address space. Also, any given processor

can use the page tables describing the virtual memory of any other processor.

As a result of this memory organization, any user process in the system can execute on any of the system's processors, even though the user process's resident memory is located on only one processor (the home site).

Since access to local memory (i.e., access by a CPU to its own memory module) is much faster than cross-processor memory access, the system is designed to run a process as much as possible on its home processor, and to move its execution to another processor only when necessary. Normally, a process's execution is moved away from its home site only to have a system request serviced which cannot be serviced at the home site.

The main processor's memory module MEM_MP holds a number of important data structures including a Process Table 30 which holds essential data regarding each of the processes in the system, a CPUSTATE data structure which contains important data on the status of the processor, and a set of USER data structures which hold data relating to the state of each process allocated to the main processor.

Each processor has a CPUSTATE data structure, which is stored in its local memory. Each processor's local memory also has an array of USER data structures, which are used to hold data relating to the processes allocated to that processor. The details and uses of these data structures are explained below.

Operating System

All of the CPUs use a modified UNIX operating system. (UNIX is a trademark of AT&T Bell Laboratories.) Full UNIX functionality is available in all the processors. To accomplish this, the UNIX kernel has been modified by dividing or replicating portions of the kernel among the MP and attached APs, and adding new interprocessor communication facilities. The interprocessor communication facilities allow requests which cannot be handled locally (e.g., on one of the APs) to be handled by another processor (e.g., the main processor MP).

For those not skilled in the art, it should be known that the term "kernel" is generally used to refer to a set of software routines, including system routines which handle most of the system's hardware dependent functions—such as disk access and other input and output operations. The UNIX kernel is typically copied from disk into the system's random access memory (e.g., the main processor's memory MEM_MP) whenever the system is restarted.

The main difference between the main processor MP and the applications processors AP1 to APn is that the main processor MP is the only processor that can perform certain system functions. For instance, the main processor is the only one which can access the system's disk storage units 22.

The primary goal for the allocation of system functions between the processors is to make the operation of each processor as autonomous as the system's hardware configuration will allow. In other words, each AP is allowed to perform as many system functions locally as is consistent with the system's hardware. This minimizes the frequency of interprocessor function calls and interprocessor memory access.

Interprocessor Busses

Another difference between the main processor MP and the applications processors AP1 to APn is that the main processor has different interprocessor bus connections than the other processors.

The system 20 includes two interprocessor busses 24 and 26. One, called the system composition bus 24, can transfer data between processors at a rate of 12 megabytes per second (12 MB/sec). The other bus, called the interprocessor bus 26, can transfer data between processors at 33.3 MB/sec. The provision of two such busses allows the faster bus to handle video tasks and other task which require high data rates, while the slower bus handles interprocessor memory requests between the main processor and one of the applications processors.

All of the applications processors, AP1 to APn and the system's Display Processor 28 are connected to both busses 24 and 26. The main processor MP is not connected to the faster bus mostly to avoid the cost of adding another high speed port to the main processor, which is already burdened with the disk controller and two terminal ports (not shown in the Figures).

Indivisible Read-Modify-Write Instructions

As will be explained in more detail below, the processors in the system must be able to perform at least one of the "indivisible read-modify-write instructions" known as test and set (TAS), or compare and swap (CAS). Basically, an "indivisible read-modify-write instruction" is an instruction which involves two steps, a test step and then a conditional data writing step, which cannot be interrupted until it is complete.

For instance, the TAS instruction can be used to test if the top bit of a specified memory location is equal to zero, and, if so, to set it equal to one. Making this instruction "indivisible" ensures that while processor AP1 performs a TAS on memory location X, no other processor can modify X. Otherwise, another processor, such as MP or AP2 could modify X after AP1 had tested X's value but before AP1 was able to set X to 1.

An indivisible compare and swap (CAS) instruction works similarly to the TAS instruction, except that the first step compares a first CPU register with a memory location, and the second step stores the value in a second CPU register into the memory location if the comparison's test criterion is satisfied.

In addition to the run queue lock, indivisible read-modify-write instructions are used for updating all data structures that could otherwise be simultaneously accessed and modified by more than one processor. In other words, each such data structure must have a corresponding lock flag and an indivisible read modify write instruction must be used to test the lock flag before the corresponding data structure is updated. Data structures, such as the USER structure, which cannot be simultaneously accessed by more than one processor do not need this lock protection. As will be understood by those skilled in the art, an example of another data structure which requires the use of a lock flag is the sleep queue for each processor.

Process Table

In the main processor's memory module there is a data structure called the process table 30. This table has one row 38 of data for each user process in the system, including the following data which are used in the present invention.

There is a priority parameter 31 which indicates the relative priority of the process. In the preferred embodiment, numerically low priority parameter values are used to indicate high process priority. User processes are assigned priority values between 127 (the lowest priority) and 25 (the highest priority), while system tasks are assigned priority values between 24 and zero.

Each process in the system is either actively running, is waiting to run, or is temporarily inactive. The processes waiting to run on each processor are placed on separate run queues. Similarly, there is a set of sleep queues for temporarily inactive processes, and actively running processes are not on any queue as long as they are running.

Each run queue is simply a linked list formed using the queue link parameter 31 in the process table 30. For each CPU there is a CPUSTATE data structure 40 which points to the row 38 of the process table 30 for the first process in its run queue. The queue link 31 for that process points to the row of the process table for the next process in the processor's run queue, and so on. The entry for the last process in each run queue has a zero in its queue link to indicate that it is at the end of the queue.

For each process, the process table 30 also includes a Home CPU parameter 33 which identifies the assigned home CPU for the process, and a Current CPU parameter 34 which identifies the current CPU on which it is running or waiting to run. The Next CPU parameter 35 is used in cross processor calls to indicate the CPU on which the process is to be run:

Finally, for each process there is a User Struc parameter 36 which points to the User Structure for the process. A separate User Structure, which is a (3072 byte) buffer, is assigned to every process for storing the state of the process, certain special parameters, and for holding a stack called the system mode stack that is used when the process performs certain system mode functions.

While the process table 30 contains other parameters used by the UNIX operating system, only those used for cross processor calls are described herein.

CPUSTATE Data Structure

FIG. 3 is a block diagram of the CPUSTATE data structure used in the preferred embodiment of the present invention. There is one CPUSTATE data structure for each processor in the system.

The CPUSTATE data structure for each processor contains the following parameters. A Run Queue Header 40*a* points to the row of the process table 30 for the first process in the processor's run queue. A somewhat simpler way to state this is to say that the Run Queue Header 40*a* points to the first process in the run queue for the corresponding processor.

The Current Proc parameter 40*b* points to the row of process table 30 for the process currently running in the processor. The Last Proc parameter 40*c* points to the row of process table 30 for the last process which ran in the processor before the current process began running.

The Current Priority parameter 40*d* is the priority value for the process currently running in the processor.

The Run Lock parameter 40*e* is a flag which is normally equal to zero. Any processor which wants to update or modify the processor's run queue is required to check this flag using a TAS instruction before proceeding. If the Run Lock flag is not zero, then some other processor is modifying the run queue and the first processor must wait until it is done. To prevent excessive bus traffic caused by such situations, if a processor finds that a run queue is locked, it is forced to wait a preselected amount of time (e.g., 0.01 milliseconds) before testing the Run Lock again.

If the Run Lock is equal to zero, the run queue is unlocked and the process performing the test sets the Run Lock before proceeding to modify the processor's run queue. When the processor is done with the run queue, it unlocks the run queue by resetting the Run Lock to zero.

The Preemption Flag parameter 40*f* is used to force the processor to look on its run queue for a process of higher priority than the currently running process. Normally, the search for a new process is performed only when the currently running process finishes or reaches a block (such as a cross processor call) which causes it to stopped, at least temporarily. The current process can be preempted, however, if the Preemption Flag 40*f* is given a nonzero value.

Toward the end of the processing of every interrupt which can interrupt a user process, the Preemption Flag 40*f* is checked. If the Preemption Flag is nonzero, a search for a higher priority process than the currently running process is initiated. If a higher priority process is found, the current process is stopped and saved, and the higher priority process is run. In any case, at the end of the preemption search, the Preemption Flag 40*f* is automatically reset to zero.

The set of interrupts which can initiate a preemption search include the interrupt generated by a processor when it puts a high priority process on the run queue of another processor, and a clock interrupt, which occurs 64 times per second.

Next, the CPUSTATE data structure contains a set of Performance Statistics 40*g* which, as described below, are used to help select to a home site for each new process created by the system 20.

The CPUSTATE data structure 40 also contains a list 40*h* of the free pages in its physical memory for use by its virtual memory management system:

Cross Processor Calls

FIG. 4 is a flow chart of the process by which a system subroutine call may cause a process to be moved from one site to another in a computer system. This is essentially a more detailed version of FIG. 2A.

For the purpose of explaining FIGS. 4 through 6, the term "the process" is used to refer to the process which has made a syscall or other subroutine call which has caused a cross processor call to be made.

Whenever a syscall (i.e., a system subroutine call) is made the system first checks to see if the syscall can be run on the current CPU of the process which made the call (box 70). If so, the syscall routine is performed locally (box 86) and, assuming that the process is running on its home CPU (box 88), it returns to the process that performed the syscall (box 90).

If the syscall cannot be performed locally, the following steps are performed. The parameter in the process table called Next CPU is set equal to a pointer to a processor (i.e., to the CPUSTATE data structure for a processor) which can perform the syscall (box 76). Then a variable called Old Priority is set equal to the process's priority (box 78) (which is obtained from the priority entry for the current process in the process table) so that this value can be restored after a context switch is performed.

Next, the process's priority is set to the highest user priority allowed by the system (box 80). This is done so that the process will be serviced as quickly as possible by the main processor.

The context switch itself is performed by calling a routine called SWITCH (box 82), which is described below in detail with reference to FIGS. 5 and 6. The process is now running on the processor pointed to by the Next CPU parameter in the process table 30. In the preferred embodiment, the Next CPU is always the main processor MP, but in other embodiments of this invention the Next CPU could be any processor in the system.

Once the context switch has been made, the process's original priority is restored (box 84) and the syscall routine is performed (86). Then the process is returned to its home cite (boxes 88 to 94). This is done by first checking to see if the current CPU is the process's Home CPU (box 88).

If the Current CPU is the process's Home CPU, no further processing is required and the routine returns (box 90). Otherwise a context switch back to the process's Home CPU is performed by setting the Next CPU parameter equal to the process's Home CPU (box 92), calling SWITCH (box 94), and then returning (box 90).

In the preferred embodiment of the invention, the restoration portion of the syscall handling routine, boxes 88 to 94, is kept simple because (1) there is only one processor to which processes are sent for handling special functions (i.e., the main processor), and (2) none of the syscall routines call other syscall routines. As a result, a process is always returned to its Home CPU after a syscall is complete.

As will be understood by those skilled in the art, in other embodiments of the invention a process might "return" to a CPU other than the process's Home CPU. In such a system a "Return CPU" parameter would have to be added to the system. In such a system, box 88 will be replaced by a query regarding whether "Current CPU=Return CPU?", and if not, the process will be SWITCHed to the Return CPU, which may or may not be the Home CPU.

FIG. 5 is a flow chart of the SWITCH routine used in the preferred embodiment of the invention to move a process from one site to another in a computer system. The first step (box 100) of this routine is to save the context of the current process by storing its registers and such in its USER data structure (see FIG. 1).

Then the run queue for the Current CPU is locked (using the the RUN LOCK 40*e* in the CPUSTATE data structure for the Current CPU) (box 104) and the LAST PROC parameter 40*c* is set equal to CURRENT PROC 40*b*. This reflects the fact that the current process will no longer be running, and hence will be the last process to have run.

Next the CURRENT PROC parameter 40*b* is set equal to a pointer to the process on the run queue with the highest priority, and the CURRENT PRIORITY parameter 40*d* is set equal to this new process's priority (box 108). Then the new CURRENT PROC is removed from the run queue (by modifying the run queue's linked list using standard programming techniques) (box 110) and the Current CPU's run queue is unlocked (box 112) by setting the RUN LOCK parameter 40*e* to zero. Finally, the new current process is started up by restoring the process's context from its USER structure and "resuming" the process (box 114).

Now that the processor which was running the user process has been set up with a new process, the SWITCH program continues by setting up the NEXT CPU to run the user process which is undergoing the context switch. These steps are performed whenever a context switch is done—i.e., only when the NEXT CPU is not the CURRENT CPU (box 117).

If the process which called SWITCH (now called LAST PROC) is done, or otherwise ready to exit (box 116), and doesn't need any further processing except to be cleaned up and removed from the system, then this clean up is done (box 118) and the SWITCH routine exits (box 122).

Otherwise, if the process (called LAST PROC) is being moved to a new CPU (i.e., its NEXT CPU is not the same as its CURRENT CPUT) (box 117) the SETRQ routine is called (box 120) to move the process identified by LAST PROC to its home site.

If the process which called SWITCH is being suspended (i.e., being put on a sleep queue) or is not switching processors for any other reason (box 117) then the SWITCH routine exits instead of calling SETRQ. Thus the SWITCH is used not only for context switching, but also for activating a new process whenever a CPU's currently running process is suspended.

FIG. 6 is a flow chart of the subroutine SETRQ which is used by the context switching method diagramed in FIG. 5. The SETRQ routine receives as a parameter a pointer to a process which needs to be put onto the run queue of its NEXT CPU, as identified by the entry in the process table 30 (FIG. 1) for that process.

The first step (box 130) of the SETRQ routine is to lock the run queue of the process's NEXT CPU, using an indivisible TAS instruction on the Run LOCK for that CPU. As indicated above, if the Run LOCK for the NEXT CPU is already locked, the system waits for a short period of time and then tries the TAS instruction again, repeating this process until the Run LOCK is unlocked by whatever process was previously using it.

Once the NEXT CPU's run queue is locked, the routine checks to see if the Current CPU is the same as the NEXT CPU (box 132). If so, the process is simply added to the run queue of the current (i.e., NEXT) CPU (box 136) and the run queue for the NEXT CPU is unlocked (box 138) by setting its Run LOCK to zero.

If the current CPU is not the same as the NEXT CPU (box 132) then the Preemption Flag of the NEXT CPU is set and the NEXT CPU is sent a software interrupt (box 134). Then the Current CPU parameter is set equal to NEXT CPU, the process is added to the run queue of the NEXT CPU (box 136) and the NEXT CPU's run queue is unlocked.

The purpose of setting the Preempt Flag and generating an interrupt for the NEXT CPU (box 134) is to force the NEXT CPU to service the process as soon as possible. This combination of steps forces the NEXT CPU to preempt the currently running process with the process added to the run queue, which has been given the highest user priority (see box 108 in FIG. 5). The reason that the interrupt is sent to the NEXT CPU before the process is added to the NEXT CPU's run queue is simply to speed up the process of transferring the process to the NEXT CPU. In effect, the NEXT CPU is forced to begin the process of looking for a new process as soon as possible, but will not actually look through its run queue for the highest priority process

therein until its run queue is unlocked several instruction cycles later.

Process Creation and Home Site Selection

Referring to FIG. 7, new processes are created in UNIX systems by a two step process: a fork (box 150) which duplicates an already existing process, and an "exec" step (box 152) which replaces the duplicated process's control program with a program from a specified file.

In systems incorporating the present invention, process creation requires an additional step: selection of a home cite for the new process. In the preferred embodiment, the selection of a home site works as follows.

First (box 154), the process inspects the HOMEMASK parameter for the new process. The HOMEMASK parameter is loaded into the new process's USER structure when the process is created from the paren process at the fork step (box 150). It is a mask that indicates which CPUs the new process can use as a home site. In particular, each bit of the HOMESITE parameter is equal to 1 if the corresponding CPU can be used as a home site, and is equal to 0 if is can't.

The remainder of the home site selection process is restricted to sites allowed by the process's HOMESITE.

Second (box 156), the system calculates the expected memory usage of the new process for each of the processors permitted by its HOMESITE parameter. The process may use less memory in some processors than in others because it may be able to share code already resident at those sites.

Third (box 158), the system picks two candidate sites: (1) the site which would have the most memory left if the process were located there; and (2) the site with the smallest average run queue size which also has enough memory to run the process.

Fourth (boxes 160–164), the system selects as the process's home site, the first candidate if its average queue size is less than the the second candidate's average queue size plus a preselected quantity QZ (a system parameter selected by the person setting up the system, typically equal to 0.5). Otherwise the second candidate site is selected as the home site.

It should be noted that the average queue size for every processor is stored in the Performance Statistics portion 40g of the processor's CPUSTATE data structure, and is updated by the system's clock routine sixty-four times per second.

Also, it can be seen that the selection of a home site is generally weighted in favor of sites with the most memory available.

Finally (boxes 166–168) the new process is moved to its new home site (if it isn't already there), by setting its HOME CPU parameter to the new site, copying its USER structure and resident memory into the memory of the home site, and putting the new process on the home site's run queue by calling SETRQ (not shown as a separate step in FIG. 7).

Memory Management

The UNIX kernel is initially copied from disk into the main processor's random access memory MEM_MP whenever the system is restarted. The other processors initially access the kernel's routines by performing nonlocal memory accesses over the syststem composition bus 24 to MEM_MP.

For purpose of efficient operation, a copy of the portions of the UNIX kernel used by each application processor is copied into local memory. This is done so that each processor can run kernel code without having to perform nonlocal memory access, which is very slow compared to local memory access.

The process by which the kernel code is copied to local memory is as follows.

FIG. 8 is a block diagram of a virtual memory management page table used in the preferred embodiment of the present invention. The use of page tables by the memory management units is well known in the prior art. However, the present invention makes special use of this table.

For each page entry in the MMU page table there is a space for storing a real (i.e., physical) memory address, a "v" flag which indicates if the real memory address is valid (i.e., it indicates whether the page is currently stored in resident memory), a read only RO flag which indicates, if enabled, that the corresponding page cannot be overwritten; a MOD flag, which is enabled if the contents of the page have been modified since the page was first allocated; and a REF flag, which is enable if the contents of the page have been accessed in any way (i.e., either read or written to) since the page was first allocated.

When the system is first started up, the MMU Page Tables in each processor contain entries for all of the UNIX kernel code, with real addresses in the main processor's memory MEM_MP. Every five seconds, a special program in the main processor writes into local memory a copy of all the kernel pages which have been referenced by each applications processor and which have not yet been copied into local memory. References to these pages are thereafter handled by reading local memory rather than the main processor's memory.

While the present invention has been described with reference to a few specific embodiments, the description is illustrative of the invention and is not to be construed as limiting the invention. Various modifications may occur to those skilled in the art without departing from the true spirit and scope of the invention as defined by the appended claims.

For instance, in other embodiments of the present invention, the system's functions could be distributed in such a way that different syscall routines might require cross processor calls to a plurality or multiplicity of different corresponding processors.

Also, in some embodiments of the present invention there could be a dynamic load balancer which would periodically compare the loads on the various processors in the system. The load balancer would be programmed to select candidate processes for being shifted to new home sites, and to transfer a selected process to a new home site if the load on its current home site becomes much heavier than the load on one or more of the other processors in the system.

What is claimed is:

1. A computer system, comprising:

a multiplicity of distinct central processing units (CPUs), each having a separate, local, random access memory means to which said CPU has direct access;

at least one interprocessor bus coupling said CPUs to said multiplicity of memory means, so that each CPU can access both its own local memory means and the memory means of the other CPUs;

run queue means coupled to said CPUs for holding a separate run queue for each of said CPUs; each said run queue holding a list of the processes waiting to run on the corresponding CPU;

process creation means in at least one of said CPUs for creating new processes, for assigning one of said CPUs as the home site of each new process, and for installing said new process in the local memory means for said home site; and

cross processor call means in each of said CPUs for temporarily transferring a specified process from its home site to another one of said CPUs, for the purpose of performing a task which cannot be performed on said home site, said cross processor call means including means for:

(a) placing said specified process on the run queue of said other CPU;

(b) continuing the execution of said specified process on said other CPU, using the memory means for said specified process's home site as the resident memory for said process and using said interprocessor bus means to couple said other CPU to said home site memory means, until a predefined set of tasks has been completed; and then

(c) upon completion of said predefined set of tasks, automatically returning said specified process to its home site by placing said specified process on the run queue of said specified process's home site, so that execution of the process will resume on said specified process's home site.

2. A computer system as set forth in claim 1, wherein said random access memory means of a first one of said CPUs includes kernel means having a predefined set of software routines for performing predefined kernel functions;

said computer system further including

memory management means coupled to said random access memory means of said CPUs, including

table means for denoting which portions of said kernel means are used by each of said CPUs other than said first CPU; and

kernel copy means, coupled to said table means, for periodically copying into the local random access memory means of each of said other CPUs said kernel portions denoted in said table means as used by said CPU but not previously copied into the local random access memory means of said CPU;

whereby the use of said interprocessor bus for accessing said kernel means is reduced by providing copies, in the local memory means of each CPU, of those portions of said kernel means actually used by each CPU.

3. A computer system as set forth in claim 1, wherein said process creation means includes means for assigning a home site priority to each new process, said home site priority being assigned a value within a predefined range of priority values;

said system further includes process selection means for selecting a process to run on a specified one of said CPUs, when the process currently running in said specified CPU is stopped, by selecting the process in said run queue for said specified CPU with the highest priority; and

said cross processor call means further includes means for

(d) assigning a specified process a higher priority than its home site priority when it is added to the run queue for a CPU other than its home site; and

(e) resetting the priority for said specified process to its home site priority when said process is added to the run queue for its home site;

whereby a process transferred to a CPU other than its home site is given increased priority to accelerate selection of the process for running on said other CPU.

4. A computer system as set forth in claim 3, wherein at least one of said CPUs includes preemption means for finding the highest priority process in its run queue, said preemption means including means for stopping the process currently running said CPU, when said highest priority process has higher priority than the process currently running in said CPU, and then running said highest priority process;

at least one of said CPUs includes interrupt means for activating said preemption means in another one of said CPUs; and

said cross processor call means further includes means for

(f) using said interrupt means in said specified process's home site CPU to activate said preemption means in a specified CPU when said specified process is added to the run queue for said specified CPU;

whereby a process transferred to a CPU other than its home site will be run immediately if its assigned priority is greater than the priorities assigned to the process currently running in said other CPU and to other processes, if any, in said run queue for said other CPU.

5. A computer system, comprising:

a multiplicity of distinct central processing units (CPUs), each having a separate, local, random access memory means to which said CPU has direct access; said CPUs having the capability of executing indivisible read modify write instructions;

at least one interprocessor bus coupling said CPUs to all of said memory means, so that each CPU can access both its own local memory means and the memory means of the other CPUs;

run queue means coupled to said CPUs for holding a separate run queue for each of said CPUs; each said run queue holding a list of the processes waiting to run on the corresponding CPU;

a run lock for each said run queue, said run lock having a first predefined value to indicate that the corresponding run queue is not in the process of being modified by any of said CPUs and is unlocked, and a value other than said first predefined value when the corresponding run queue is being modified by one of said CPUs and is therefore locked;

run queue updating means coupled to said CPUs for adding or removing a specified process from a specified run queue, said run queue updating means including means for:

(a) locking said specified run queue by

(a,1) using an indivisible read modify write instruction to test the value of the run lock for said specified run queue and, if said run lock value indicates that said specified run queue is unlocked, to set said run lock to a value which

indicates that said specified run queue is locked; and

(a,2) if the test in step (a,1) determines that said run queue is locked, performing step (a,1) again after a predefined delay, until the test in step (a1) determines that said run queue is unlocked;

(b) adding or removing a specified process from said specified run queue; and

(c) unlocking said specified run queue by setting said run lock for said specified run queue to said first predefined value;

process creation means in at least one of said CPUs for creating new processes, for assigning one of said CPUs as the home site of each new process, and for installing said new process in the local memory means for said home site; and

cross processor call means in each of said CPUs for temporarily transferring a specified process from its home site to a specified one of said other CPUs, for the purpose of performing a task which cannot be performed on said home site, said cross processor call means including means for:

(a) using said run queue updating means to add said specified process to the run queue of said specified CPU;

(b) continuing the execution of said specified process on said specified CPU, using the memory means for said specified process's home site as the resident memory for said process and using said interprocessor bus means to couple said specified CPU to said home site memory means, until a predefined set of tasks has been completed; and then

(c) upon completion of said predefined set of tasks, automatically returning said specified process to its home site by using said run queue updating means to add said specified process to the run queue of said specified process's home site, so that execution of the process will resume on said specified process's home site.

6. A computer system as set forth in claim 5, wherein said process creation means includes means for assigning a home site priority to each new process, said home site priority being assigned a value within a predefined range of priority values;

said system further includes process selection means for selecting a process to run on a specified one of said CPUs, when the process currently running in said specified CPU is stopped, by selecting the process in said run queue for said specified CPU with the highest priority; and

said cross processor call means further includes means for

(d) assigning a specified process a higher priority than its home site priority when it is added to the run queue for a CPU other than its home site; and

(e) resetting the priority for said specified process to its home site priority when said process is added to the run queue for its home site;

whereby a process transferred to a CPU other than its home site is given increased priority to accelerate selection of the process for running on said other CPU.

7. A computer system as set forth in claim 6, wherein at least one of said CPUs includes preemption means for finding the highest priority process in its run queue, said preemption means including means for stopping the process currently running said CPU, when said highest priority process has higher priority than the process currently running in said CPU, and then running said highest priority process;

at least one of said CPUs includes interrupt means for activating said preemption means in another one of said CPUs; and

said cross processor call means further includes means for

(f) using said interrupt means in said specified process's home site CPU to activate said preemption means in a specified CPU when said specified process is added to the run queue for said specified CPU;

whereby a process transferred to a CPU other than its home site will be run immediately if its assigned priority is greater than the priorities assigned to the process currently running in said other CPU and to other processes, if any, in said run queue for said other CPU.

8. A computer system, comprising:

a multiplicity of distinct central processing units (CPUs), each having a separate, local, random access memory means to which said CPU has direct access; said CPUs having the capability of executing indivisible read modify write instructions;

at least one interprocessor bus coupling said CPUs to all of said memory means, so that each CPU can access both its own local memory means and the memory means of the other CPUs;

process creation means coupled to said CPUs for creating new processes, for assigning one of said CPUs as the home site of each new process, for installing said new process in the local memory means for said home site, and for assigning a home site priority to each new process, said home site priority being assigned a value within a predefined range of priority values;

run queue means coupled to said CPUs for holding a separate run queue for each of said CPUs; each said run queue holding a list of the processes waiting to run on the corresponding CPU;

process table means coupled to said CPUs for retaining information regarding every process running or otherwise in existence in said system, including for each said process

a HOME CPU parameter which indicates the home site of said process;

a CURRENT CPU parameter which indicates the current CPU on which said process is running, waiting to run, or otherwise residing; and

a PRIORITY parameter indicative of the priority of said process;

cpustate table means for storing information regarding each said CPU, including:

a run queue header identifying the run queue for said CPU;

a current process parameter identifying the process currently running in said CPU;

a last process parameter identifying the process which was run prior to the process currently running in said CPU; and

a run lock parameter which is given a first predefined value to indicate that the corresponding run queue is not in the process of being modified by any of said CPUs and is unlocked, and a value

other than said first predefined value when the corresponding run queue is being modified by one of said CPUs and is therefore locked; and

run queue updating means coupled to said CPUs for adding or removing a specified process from a specified run queue, said run queue updating means including means for:

(a) locking said specified run queue by

(a,1) using said indivisible read modify write instruction to test the value of the run lock for said specified run queue and, if said run lock value indicates that said specified run queue is unlocked, to set said run lock to a value which indicates that said specified run queue is locked; and

(a,2) if the test in step (a,1) determines that said run queue is locked, performing step (a,1) again after a predefined delay, until the test in step (a1) determines that said run queue is unlocked;

(b) adding or removing a specified process from said specified run queue;

(c) unlocking said specified run queue by setting said run lock for said specified run queue to said first predefined value; and

(d) updating said run queue header, current process and last process parameters of the cpustate table means for the CPU corresponding to said specified run queue to reflect the current status of said CPU;

process selection means in at least one of said CPUs for selecting a process to run on a specified one of said CPUs when the process currently running in said specified CPU is stopped, including means for selecting the process in said run queue for said specified CPU with the highest priority and means for initiating the running of said selected process in said specified CPU; and

preemption means in each CPU for finding the highest priority process in its run queue, said preemption means including means for stopping the process currently running in said CPU, when said highest priority process has higher priority than the process currently running in said CPU, and then running said highest priority process;

interrupt means in each said CPU for activating said preemption means in a specified one of the other CPUs; and

cross processor call means in each of said CPUs for temporarily transferring a specified process from its home site to a specified one of said other CPUs, for the purpose of performing a task which cannot be performed on said home site, said cross processor call means including means for:

(a) using said run queue updating means to add said specified process to the run queue of said specified CPU;

(b) assigning said specified process a higher priority than its home site priority when it is added to said run queue for said specified other CPU;

(c) using said interrupt means in said specified process's home site CPU to activate said preemption means in said specified CPU when said specified process is added to said run queue for said specified CPU, so that said specified process will be preempt the process currently running in said specified CPU;

(d) continuing the execution of said specified process on said specified CPU, using the memory means for said specified process's home site as the resident memory for said process and using said interprocessor bus means to couple said specified CPU to said home site memory means, until a predefined set of tasks has been completed; and then

(e) upon completion of said predefined set of tasks, automatically returning said specified process to its home site by using said run queue updating means to add said specified process to the run queue of said specified process's home site, so that execution of the process will resume on said specified process's home site; and

(f) resetting the priority for said specified process to its home site priority when said process is added to said run queue for its home site.

9. A method of running a multiplicity of processes in a computer system, comprising the steps of:

providing a computer system having

(1) a multiplicity of distinct central processing units (CPUs), each having a separate, local, random access memory means to which said CPU has direct access; said CPUs having the capability of executing indivisible read modify write instructions; and

(2) at least one interprocessor bus coupling said CPUs to all of said memory means, so that each CPU can access both its own local memory means and the memory means of the other CPUs;

wherein at least one of said CPUs is capable of performing one or more tasks that at least one of said other CPUs cannot perform;

generating a run queue data structure for holding a separate run queue for each of said CPUs; each said run queue holding a list of the processes waiting to run on the corresponding CPU;

providing run queue updating means for adding or removing a specified process from a specified run queue;

creating new processes, as the need arises, including the step of assigning one of said CPUs as the home site of each new process, and installing said new process in the local memory means for said home site; and

when one of said processes needs to perform a task which cannot be performed on its home site, performing a cross processor call to temporarily transfer said process from its home site to a specified one of said other CPUs which is able to perform said task, be performing the steps of:

(a) using said run queue updating means to add said process to the run queue of said specified CPU;

(b) continuing the execution of said process on said specified CPU, using the memory means for said process's home site as the resident memory for said process and using said interprocessor bus means to couple said specified CPU to said home site memory means, until a predefined set of tasks has been completed; and then

(c) upon completion of said predefined set of tasks, automatically returning said specified process to its home site by using said run queue updating means to add said process to the run queue of said process's home site, so that execution of the process will resume on said process's home site.

10. A method as set forth in claim 9, further including the step of generating a run lock flag for each said run queue, said run lock having a first predefined value to indicate that the corresponding run queue is not in the process of being modified by any of said CPUs and is unlocked, and a value other than said first predefined value when the corresponding run queue is being modified by one of said CPUs and is therefore locked;

wherein said step of providing run queue updating means includes providing run queue updating means for adding or removing a specified process from a specified run queue by performing the steps of:

(a) locking said specified run queue by

(a,1) using said indivisible read modify write instruction to test the value of the run lock for said specified run queue and, if said run lock value indicates that said specified run queue is unlocked, to set said run lock to a value which indicates that said specified run queue is locked; and

(a,2) if the test in step (a,1) determines that said run queue is locked, performing step (a,1) again after a predefined delay, until the test in step (a1) determines that said run queue is unlocked;

(b) adding or removing a specified process from said specified run queue; and

(c) unlocking said specified run queue by setting said run lock for said specified run queue to said first predefined value.

11. A method as set forth in claim 9, wherein said step of creating new processes includes assigning a home site priority to each new process, said home site priority being assigned a value within a predefined range of priority values;

said method further including the step of selecting a process to run on a specified one of said CPUs, when the process currently running in said specified CPU is stopped, by selecting the process in said run queue for said specified CPU with the highest priority; and

said step of performing a cross processor call further includes the steps of

(d) assigning a specified process a higher priority than its home site priority when it is added to the run queue for a CPU other than its home site; and

(e) resetting the priority for said specified process to its home site priority when said process is added to the run queue for its home site;

whereby a process transferred to a CPU other than its home site is given increased priority to accelerate selection of the process for running on said other CPU.

12. A method as set forth in claim 11, wherein said step of performing a cross processor call further includes the step of

(f) generating a preemption interrupt in said specified CPU when said specified process is added to the run queue for said specified CPU;

said method further including the step of responding to a preemption interrupt in a specified CPU by:

(a) finding the highest priority process in the run queue of said specified CPU; and

(b) stopping the process currently running in said CPU, when said highest priority process has higher priority than the process currently running in said CPU, and then running said highest priority process;

whereby a process transferred to a CPU other than its home site will be run immediately if its assigned priority is greater than the priorities assigned to the process currently running in said other CPU and to other processes, if any, in said run queue for said other CPU.

13. A method as set forth in claim 9, further including the steps of:

providing a predefined set of kernel routines in said local memory means of a first one of said CPUs;

denoting, in a predefined data structure, which of said kernel routines are used by each of said CPUs other than said first CPU; and

periodically copying into the local random access memory means of each of said other CPUs said kernel routines denoted in said predefined data structure as used by said CPU but not previously copied into the local random access memory means of said CPU;

whereby the use of said interprocessor bus for accessing said kernel routines is reduced by providing copies, in the local memory means of each CPU, of those kernel routines actually used by each CPU.

* * * * *